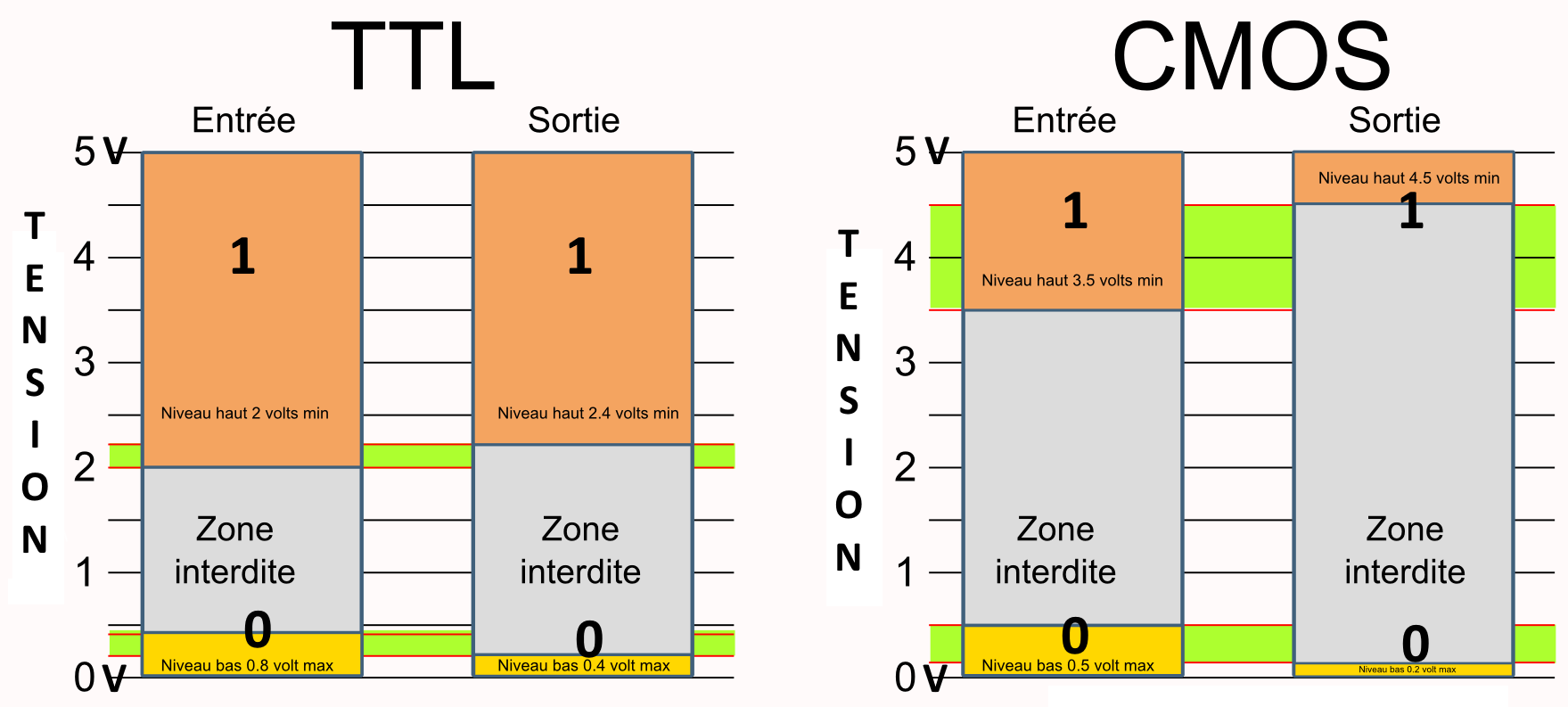
Logique booléenne

1. Introduction :

Nous avons vu précédemment que les systèmes numériques étant basés sur des transistors. Ces circuits manipulaient des **signaux logiques ( qui ne peuvent prendre que 2 valeurs : 0 ou 1**). En fait, derrière ces valeurs 0 ou 1, se cachent des niveaux de tension (légèrement différents la technologie de construction utilisée).



A partir de ces composants de bases que sont les transistors, on construit les microprocesseurs qui sont "l'évolution ultime du transistor) mais plus directement les circuits logiques de base (porte ET, OU, NON ET, circuit mémoires) .. qui constituent eux-mêmes les éléments de base des microprocesseurs et autres composants intégrés, optimisés ("microcontrôleur", FPGA, SOC…) utilisés par exemple dans vos smartphone.

Nous allons dans ce chapitre étudier un peu plus la théorie de base de ces circuits logiques : la **logique booléenne**.

**Petit historique :**

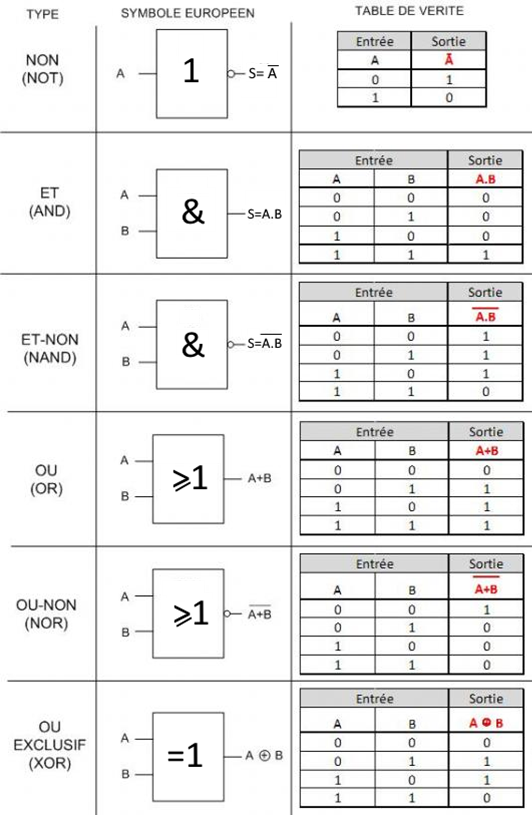
Bien que le Binaire soit apparu en 3000 avant J.C. en Chine, le terme booléen vient de Georges Boole qui sort ' Mathematical Analysis of Logic' en 1847, et pose les bases du calcul.

Il fallut attendre 1930 pour que Claude Shannon démontre qu'avec des interrupteurs fermés pour 'vrai' et ouvert pour 'faux', on peut effectuer des opérations logiques en associant le nombre 1 pour vrai et 0 pour faux. C'est lui qui va populariser le mot 'bit' créé par John Tukey.

1. Les portes de base

La logique booléenne repose sur 3 opérations de base (**NON, OU , ET**). On utilise également 3 opérations "dérivés" **NON ET** (NAND) , **NON OU**( NOR) et **OU EXCLUSIF** (XOR).

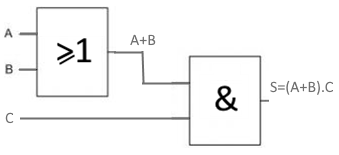
Leur comportement est donné dans le tableau page suivante, décrit par la table de vérité ( état de la grandeur de sortie en fonction de l'état des grandeurs d'entrée). On donne également le symbole de européen de la porte logique ( le symbole et le schéma est donné à titre indicatif).



**Remarques :**

* les fonctions NON-ET et NON-OU sont des fonctions universelles car elles permettent la réalisation des trois opérateurs élémentaires NON, ET et OU.
* La porte NAND (NON-ET) est la porte de base de l'électronique : quasiment tout circuit d'électronique numérique peut être réalisé à partir de NAND.
* Une expression logique peut évidemment être composée de plusieurs de ces opérateurs de base, et on peut représenter ceci par une équation ou par un schéma.

Ex : S=(A+B).C



1. Propriétés de l'algèbre de Boole

Les opérateurs ET (noté . dans une équation logique) et OU (noté +) respectent les mêmes d'associativité , de commutativité et de priorité que respectivement la multiplication et l'addition.

Associativité : **a.(b.c) = (a.b).c = a.b.c**  et **a + (b + c) = (a + b) + c = a + b + c**

Commutativité : **a.b = b.a**  et **a + b = b + a**

Priorité : a.b+c est équivalent à (a.b)+c , l'opération ET est prioritaire par rapport à l'opération OU.

D'autres propriétés sont propres à l'algèbre de Boole :

Éléments neutres : **a + 0 = a**  et **a⋅1 = a**

Eléments absorbants : **a + 1 = 1** et **a.0 = 0**

Complémentarité **: 1** et **0**

Idempotence : **a + a = a** et **a . a =**  a

Absorption : **a + a.b = a.(1+b) = a.1** =a

et  **a+b = a+b.1= a+b(a|+a) = a+ a|. b +a.b = a+ a|.b**

Distributivité : **a.(b+c)=a.b + a.c** et **a+(b⋅c)=(a + b)⋅(a + c)**

Théorèmes de DE MORGAN :  **et**

("on distribue aussi la barre qui modifie le symbole de l'opération")

Quelques exemples :

En utilisant les propriétés précédentes, simplifier les expressions suivantes :

b . (a + a|) = b . 1 = b

a.b = a|. b| = 1

a.a + a.b| + b.a + b.b| = a + a.b|+ a.b = a(1 + b|+ b) = a.1 = a

1. Table de vérité

On peut représenter le comportement d'une expression logique par sa table de vérité, où pour chaque combinaison des grandeurs d'entrée, on donne la valeur de sortie.

Point important : une expression, circuit à **n entrées** logiques d'entrée aura : **2n combinaisons différentes.**

Exemple : table de vérité de : S =

Ici, 3 entrées -> **23=8 combinaisons**

Pour être sûr de ne pas oublier de combinaisons, on les dénombre en comptant en binaire

Pour faciliter le remplissage de la table de vérité, on aurait pu rajouter des colonnes pour les blocs constituant l'expression : ici : a.b puis

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | S |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Dans le sens inverse, on peut donner l'expression de la sortie à partir de sa table de vérité. La sortie est le OU logique entre les combinaisons d'entrée où S vaut 1. ( ou ligne de 0 donne )

Exemple :

On a donc :

S=

Ou plus simple : (lectures des lignes à 0 )

donc

|  |  |  |
| --- | --- | --- |
| a | b | S |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. Booléens en programmation

Python utilise le mot clé **True** pour désigner le niveau 1 logique et **False** pour 0. (**true/false** pour JavaScript)

Par extension, la chaine vide '', le nombre 0 sont considérés comme False. N'importe quel autre nombre ou chaine de caractère est considéré comme True.

On vous donne ci-dessous quelques exemples de syntaxe des opérateurs logiques suivant le langage.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Python | JavaScript | PHP |
| NON | **not a** | **!a** | **!$a** |
| ET | **a and b** | **a && b** | **$a && $b** ou **$a AND $b** |
| OU | **a or b** | **a || b** | **$a || $b** ou **$a OR $b** |
| OU exclusif | **a^b** | **a^b** | **$a XOR $b** |

**Remarques :**

Python et JavaScript évaluent les expressions logiques de manière paresseuses (on parle de lazy evaluation). Les opérateurs sont de type court-circuit :

* OR (|| pour js) n’évalue le deuxième argument que si le premier est faux. (si le premier argument est 1 , forcément le résultat vaut 1)
* AND (&& pour js) n’évalue le deuxième argument si le premier est vrai.