Introduction à l'Electronique Numérique

Jean-Christophe Le Lann

Cours 1A

24 septembre 2026

ENSTA INSTITUT POLYTECHNIQUE DE PARIS

1– Algèbre de Boole

L'Espace de Boole $\mathbb{B}$

L'algèbre booléenne [1] opère sur un ensemble à deux éléments :

$$\mathbb{B} = \{0, 1\}$$

où 0 représente **Faux** et 1 représente **Vrai**.

Variable booléenne simple

- Une variable x qui ne peut prendre que deux valeurs.
- $x \in \{0, 1\}$
- Exemple : *Un interrupteur est ouvert (0) ou fermé (1).*

Variable booléenne générale (vecteur)

- Un vecteur de n bits : $X = (x_{n-1}, \ldots, x_0)$
- $X \in \mathbb{B}^n$
- Exemple : *Un octet (8 bits) représentant un état.*

1. L'algèbre de Boole est une algèbre au sens de structure algébrique, mais pas au sens d'algèbre sur un corps $\mathbb{K}$.

Définition

Une **fonction booléenne** à n variables est une application :

$$f : \mathbb{B}^n \to \mathbb{B}$$

Elle associe à chaque combinaison d'entrées une unique sortie binaire.

Représentation

- **Table de vérité** : énumère les 2^n cas possibles.
- **Expression algébrique** : combinaison de $+, \cdot, \bar{}$.

Exemple : $f(x, y) = x \cdot \bar{y} + \bar{x} \cdot y$ (XOR)

x	y	$f(x, y)$
0	0	0
0	1	1
1	0	1
1	1	0

2– Opérations

On utilise usuellement les symboles $+$ pour le **OU** et $\cdot$ (ou $*$) pour le **ET**.

Négation (NON)

$\bar{x}$ ou $\neg x$

x	$\bar{x}$
0	1
1	0

Conjonction (ET)

$x \cdot y$ ou $x * y$

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Disjonction (OU)

$x + y$

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Opérateurs Dérivés

Au-delà des trois opérations de base $(+, \cdot, \bar{})$, on définit des **opérateurs dérivés** très utilisés en logique et en électronique.

Nom	Symbole	Expression en base
OU exclusif (XOR)	$x \oplus y$	$x \cdot \bar{y} + \bar{x} \cdot y$
NAND	$x \uparrow y$	$\overline{x \cdot y}$
NOR	$x \downarrow y$	$\overline{x + y}$
Équivalence (XNOR)	$x \Leftrightarrow y$	$x \cdot y + \bar{x} \cdot \bar{y}$
Implication	$x \Rightarrow y$	$\bar{x} + y$

Idée clé

Tous ces opérateurs se ramènent aux trois opérations fondamentales.

Le OU Exclusif ($x \oplus y$)

Définition

« x ou y, mais pas les deux » — noté $x \oplus y$.

$$x \oplus y \;=\; x \cdot \bar{y} + \bar{x} \cdot y$$

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Propriétés utiles :

- $x \oplus x = 0$
- $x \oplus \bar{x} = 1$
- $x \oplus 0 = x$
- $x \oplus 1 = \bar{x}$
- **Associatif et commutatif**

Application : addition binaire, détection de parité.

NAND et NOR : les Portes Universelles

NAND ($x \uparrow y$)

$$x \uparrow y = \overline{x \cdot y}$$

x	y	$x \uparrow y$
0	0	1
0	1	1
1	0	1
1	1	0

NOR ($x \downarrow y$)

$$x \downarrow y = \overline{x + y}$$

x	y	$x \downarrow y$
0	0	1
0	1	0
1	0	0
1	1	0

Universalité

Toute fonction booléenne peut être réalisée en n'utilisant **que des NAND** ou **que des NOR**.

L'Équivalence ($x \Leftrightarrow y$)

Définition

« x si et seulement si y » — notée $x \Leftrightarrow y$, également écrite $x \leftrightarrow y$ ou XNOR.

$$x \Leftrightarrow y \;=\; x \cdot y + \bar{x} \cdot \bar{y}$$

x	y	$x \Leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

Points à retenir :

- Vraie quand x et y sont **identiques**.
- C'est la **négation du XOR**.
- $x \Leftrightarrow y = \overline{x \oplus y}$

Lien avec l'implication :

$$(x \Rightarrow y) \cdot (y \Rightarrow x) = x \Leftrightarrow y$$

L'Implication $(x \Rightarrow y)$: Définition

Définition

« Si x alors y » — notée $x \Rightarrow y$ ou $x \to y$.

$$x \Rightarrow y \;=\; \bar{x} + y$$

x	y	$x \Rightarrow y$
0	0	1
0	1	1
1	0	**0**
1	1	1

Un seul cas rend l'implication fausse :

$$x = 1 \quad \text{et} \quad y = 0$$

C'est le seul **contre-exemple** possible : l'antécédent est vrai mais la conclusion ne l'est pas.

Reformulation : $x \Rightarrow y = \overline{x \cdot \bar{y}}$ *(négation d'un contre-exemple).*

Implication : pourquoi est-elle vraie quand $x = 0$?

La difficulté

Intuitivement, on se demande pourquoi $0 \Rightarrow 0$ et $0 \Rightarrow 1$ valent 1.

Clé : l'implication est une **contrainte conditionnelle**.

$$x \Rightarrow y \quad \text{signifie :} \quad \text{« \textbf{si} } x \text{ est vrai, \textbf{alors} } y \text{ doit être vrai. »}$$

- Si $x = 1$: la contrainte s'applique. Il faut $y = 1$, sinon elle est **violée**.
- Si $x = 0$: la contrainte **ne s'applique pas**. Elle n'est donc **pas violée**, donc elle est **vraie par défaut** (*vacuité*).

Analogie météo

« S'il pleut, alors je prends mon parapluie. »

Cette règle n'est **violée que dans un seul cas** : il pleut **et** je ne prends pas de parapluie.

Implication : Précaution d'Usage et Table Météo

Précaution d'usage

L'énoncé *« S'il pleut, je prends mon parapluie »* doit être compris comme une **règle que je m'engage à suivre**, et non comme une **causalité physique**. De plus, il **ne dit rien** sur mon comportement quand il ne pleut pas.

Il pleut x	**Parapluie** y	**Règle** $x \Rightarrow y$	**Interprétation**
1	1	1	Règle respectée
1	**0**	**0**	**Règle violée**
0	1	1	Pas de pluie — libre de mes choix
0	0	1	Pas de pluie — libre de mes choix

Contraposition :

$x \Rightarrow y \equiv \bar{y} \Rightarrow \bar{x}$

« Si je n'ai pas de parapluie, alors il ne pleut pas. »

Identités remarquables :

$$(0 \Rightarrow y) = 1$$
$$(1 \Rightarrow y) = y$$
$$(x \Rightarrow 1) = 1$$
$$(x \Rightarrow 0) = \bar{x}$$

Autres Formes d'Implication

Règle de base : *« S'il pleut, je prends un parapluie. »* $(x \Rightarrow y)$

Réciproque

$y \Rightarrow x$
« Si je prends un parapluie, alors il pleut. »
Non équivalente.

Réciproque de la contraposée

$\bar{x} \Rightarrow \bar{y}$
« S'il ne pleut pas, je ne prends pas de parapluie. »
Non équivalente.

Contraposée

$\bar{y} \Rightarrow \bar{x}$
« Si je ne prends pas de parapluie, alors il ne pleut pas. »
Équivalente à la règle de base.

Équivalence

$(x \Rightarrow y) \cdot (y \Rightarrow x) = x \Leftrightarrow y$
« Je prends un parapluie si et seulement s'il pleut. »

Piège classique

Confondre une implication avec sa réciproque est l'erreur logique la plus fréquente.

Récapitulatif des Opérateurs Booléens

x	y	$\cdot$	$+$	$\oplus$	$\Leftrightarrow$	$\Rightarrow$	$\uparrow$
0	0	0	0	0	1	1	1
0	1	0	1	1	0	1	1
1	0	0	1	1	0	0	1
1	1	1	1	0	1	1	0

Opérateurs de base :

- ET ($\cdot$), OU ($+$), NON ($^-$)

Opérateurs dérivés :

- XOR, XNOR, implication, NAND, NOR

16 fonctions booléennes à 2 variables existent[2] *— celles-ci sont les plus utilisées.*

2. Deux variables d'où 4 lignes et 2 choix pour chacune des lignes : $2 \times 2 \times 2 \times 2 = 2^4 = 16$

Priorité des Opérations et Parenthésage

Comme en arithmétique, il existe un ordre de priorité implicite.

Ordre de priorité (du plus fort au plus faible)

1. **Négation** : $\bar{x}$
2. **Conjonction** : $x \cdot y$
3. **Disjonction** : $x + y$

Exemple :

$$\bar{x} \cdot y + z = \big((\bar{x}) \cdot y\big) + z$$

En cas de doute, les parenthèses restent la meilleure amie du logicien.

3– Propriétés Algébriques

Commutativité et Associativité

Les opérations $+$ et $\cdot$ possèdent des propriétés similaires à l'arithmétique classique.

Commutativité

L'ordre des opérandes n'influence pas le résultat.

$$x + y = y + x$$
$$x \cdot y = y \cdot x$$

Associativité

L'ordre d'évaluation (groupement) n'influence pas le résultat.

$$(x + y) + z = x + (y + z)$$
$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

Particularité de l'algèbre de Boole

Contrairement à l'arithmétique, le OU est distributif sur le ET **et réciproquement**.

ET sur OU (classique)

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

OU sur ET (surprenant !)

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

Cette double distributivité est une signature de l'algèbre de Boole.

4– Théorèmes

Ces théorèmes permettent de simplifier les expressions logiques.

Éléments Neutres

$$x + 0 = x$$
$$x \cdot 1 = x$$

Éléments Absorbants

$$x + 1 = 1$$
$$x \cdot 0 = 0$$

Idempotence

$$x + x = x$$
$$x \cdot x = x$$

Complémentation

$$x + \bar{x} = 1$$
$$x \cdot \bar{x} = 0$$

Lois d'Absorption

Définition

L'absorption permet d'éliminer des termes redondants dans une expression.

$$x + x \cdot y = ?$$
$$x \cdot (1 + y) = x$$

Intuition :

Si x est vrai, l'expression est vraie quel que soit y.

Exemple d'application

$$x + x \cdot y + x \cdot z = x \cdot (1 + y + z) = x$$

Principe

Toute identité booléenne reste vraie si l'on échange simultanément :

$$+ \longleftrightarrow \cdot \qquad \text{et} \qquad 0 \longleftrightarrow 1$$

Cela explique pourquoi les théorèmes vont souvent par paires :

Identité	Identité duale
$x + 0 = x$	$x \cdot 1 = x$
$x + 1 = 1$	$x \cdot 0 = 0$
$x + x = x$	$x \cdot x = x$
$\overline{x + y} = \bar{x} \cdot \bar{y}$	$\overline{x \cdot y} = \bar{x} + \bar{y}$

Lois de De Morgan

Ces lois sont fondamentales pour la manipulation des expressions booléennes et la conception de circuits logiques.

Première Loi de De Morgan

La négation d'une disjonction est la conjonction des négations.

$$\overline{x + y} = \bar{x} \cdot \bar{y}$$

Deuxième Loi de De Morgan

La négation d'une conjonction est la disjonction des négations.

$$\overline{x \cdot y} = \bar{x} + \bar{y}$$

En résumé : l'inverse d'une somme n'est pas la somme des inverses, mais le produit des inverses.

5– Application

Exemple : Vote Majoritaire à 3 Personnes

Énoncé : Trois personnes votent. La décision est adoptée si **au moins deux** votent OUI.

Variables : $a, b, c \in \{0, 1\}$ (1 = OUI).

Forme SOP (lignes où $f = 1$) :

$$f = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

a	b	c	f
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

Simplification par les Théorèmes

Expression initiale (SOP) :

$$f = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

Étape 1 — Idempotence (on duplique abc) :

$$f = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc + abc$$

Étape 2 — Regroupement :

$$f = (\bar{a}bc + abc) + (a\bar{b}c + abc) + (ab\bar{c} + abc)$$

Étape 3 — Distributivité ($x \cdot y + x \cdot \bar{y} = x$) :

$$f = bc + ac + ab$$

Résultat : $f = ab + ac + bc$

Forme simplifiée bien plus économique à implémenter.

6– Décomposition de Shannon et Formes Canoniques

Théorème — Expansion de Shannon

Toute fonction booléenne $f : \mathbb{B}^n \to \mathbb{B}$ peut être décomposée par rapport à une variable x_i :

$$f(\dots, x_i, \dots) = x_i \cdot f_{x_i=1} + \bar{x_i} \cdot f_{x_i=0}$$

où $f_{x_i=1}$ et $f_{x_i=0}$ sont les **cofacteurs** de f par rapport à x_i.

Forme duale (par rapport à $+$) :

$$f = (x_i + f_{x_i=0}) \cdot (\bar{x_i} + f_{x_i=1})$$

Interprétation :

- On **factorise** par x_i.
- On évalue f dans les deux cas $x_i = 1$ et $x_i = 0$.
- Chaque cofacteur a une variable en moins.
- BDD en CAO : *binary decision diagrams*.

Exemple : XOR

$f(x, y) = x \oplus y$

Décomposition par rapport à x :

$$f_{x=1} = \bar{y}$$
$$f_{x=0} = y$$

$$f = x \cdot \bar{y} + \bar{x} \cdot y$$

SOP (*Sum of Products*)

Somme de **mintermes** (produits où chaque variable apparaît une fois).

$$f = \bar{x} \cdot y + x \cdot \bar{y}$$

Construite à partir des lignes où $f = 1$.

POS (*Product of Sums*)

Produit de **maxtermes** (sommes où chaque variable apparaît une fois).

$$f = (x + y) \cdot (\bar{x} + \bar{y})$$

Construite à partir des lignes où $f = 0$.

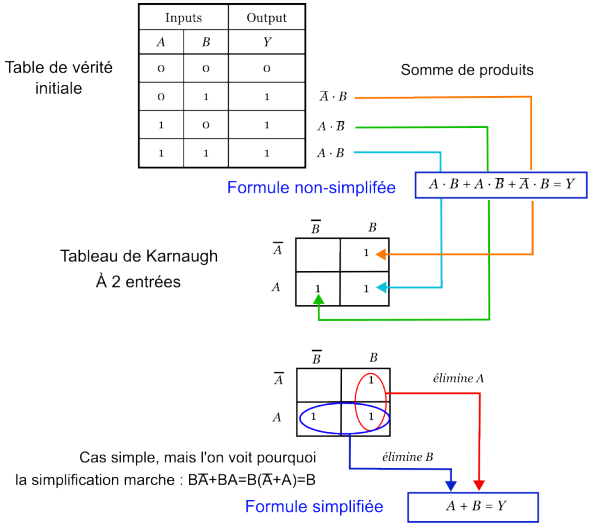
Lien avec la table de vérité

Toute fonction booléenne admet une unique forme canonique SOP et une unique forme canonique POS.

7– Tableau de Karnaugh

Tableau de Karnaugh : intuition

- **objectif** : simplifier une formule booléenne.
- **moyen** : disposition spatiale *particulière*.



Distance de Hamming

Pour deux vecteurs booléens $u, v \in \mathbb{B}^n$, on définit :

$$d_H(u,v) = \left|\{i \in \{0,\ldots,n-1\} : u_i \neq v_i\}\right|$$

C'est le nombre de bits qui diffèrent entre u et v.

Définition — Adjacence

Deux cases du tableau de Karnaugh sont dites **adjacentes** si et seulement si : $d_H(u,v) = 1$ c'est-à-dire qu'elles ne diffèrent que par **une seule variable**.

Justification : si deux cases adjacentes ne diffèrent que par x, leurs mintermes sont $x \cdot m$ et $\bar{x} \cdot m$, et :

$$x \cdot m + \bar{x} \cdot m = (x + \bar{x}) \cdot m = m$$

La variable x **disparaît**.

Conséquence topologique :
Le tableau est **torique** — les bords opposés sont adjacents.

Exemple : 000 *et* 100 *sont adjacents (*$d_H = 1$*).*

Problème

Sur un tableau ordonné en binaire naturel, les cases voisines ne sont pas nécessairement adjacentes au sens de Hamming.

Solution : Code de Gray

Ordonner les lignes et colonnes selon la suite :

$$00 \to 01 \to 11 \to 10$$

Deux entrées consécutives diffèrent d'**un seul bit** ($d_H = 1$).

$a \backslash bc$	00	01	11	10
0				
1				

Chaque case = un minterme.

Vérification : 00 et 01 sont adjacents, 01 et 11 aussi, et ainsi de suite. La suite Gray garantit que toutes les cases voisines du tableau sont adjacentes au sens de Hamming.

Exemple : Tableau de Karnaugh à 3 Variables

Fonction : $f(a,b,c) = \sum m(1,3,5,7)$

$a\backslash bc$	00	01	11	10
0	0	**1**	**1**	0
1	0	**1**	**1**	0

Colonnes en code Gray : $00, 01, 11, 10$.

Résolution :

- On regroupe les 4 **1** en un bloc 2×2.
- Variables qui **changent** : a et b.
- Variable qui **reste** : $c = 1$.
- (Erreur dans le slideware distribué).

Résultat

$$f = c$$

De 4 mintermes à un seul littéral.

Regroupements : Règles Formelles

Regroupement valide

Un regroupement est un sous-ensemble S de cases contenant des 1, tel que :

- $|S| = 2^k$ pour un certain $k \in \mathbb{N}$;
- S forme un **sous-cube** : produit cartésien de k paires d'entrées adjacentes.

Interprétation

Un sous-cube de taille 2^k correspond à l'élimination de k variables par adjacence successive.

Démarche :

1. Remplir le tableau depuis la table de vérité.
2. Identifier les **plus grands sous-cubes**.
3. Vérifier que tous les 1 sont couverts.
4. Lire le terme associé à chaque sous-cube.

Limites

- Pratique jusqu'à 4 variables.
- Difficile dès 5.
- Au-delà : **Quine-McCluskey** ou **BDD**.

8– Du booléen au circuit

Portes élémentaires

Chaque opération booléenne correspond à une **porte logique** physique, *pour peu qu'elle ait été conçue par le fondeur* (au niveau transistor).

Opération	Symbole	Porte logique
Négation	$\bar{x}$	NOT
Conjonction	$x \cdot y$	AND2
Disjonction	$x + y$	OR2
Disjonction	$x + y + z$	OR3

Portes universelles

Les portes **NAND** et **NOR** sont dites *universelles* : toute fonction booléenne peut être réalisée à partir de l'une d'elles uniquement.

L'algèbre de Boole est le langage mathématique de l'électronique numérique.

Portes logiques : symboles usuels

Porte OUI (YES)			entrée 0 1 — sortie 0 1
Porte NON (NO)			entrée 0 1 — sortie 1 0
Porte ET (AND)			entrées 0 0 / 0 1 / 1 0 / 1 1 — sortie 0 0 0 1
Porte OU (OR)			entrées 0 0 / 0 1 / 1 0 / 1 1 — sortie 0 1 1 1
Porte OU exclusif (XOR)			entrées 0 0 / 0 1 / 1 0 / 1 1 — sortie 0 1 1 0
Porte NON-ET (NAND)			entrées 0 0 / 0 1 / 1 0 / 1 1 — sortie 1 1 1 0
Porte NON-OU (NOR)			entrées 0 0 / 0 1 / 1 0 / 1 1 — sortie 1 0 0 0

Définition (tentative) — Porte logique

Une **porte logique** à k entrées est un triplet :

$$g = (f_g, \tau_g, A_g)$$

où :

- $f_g : \mathbb{B}^k \to \mathbb{B}$ est la **fonction réalisée** ;
- $\tau_g \geq 0$ est le **délai de propagation** (temps) ;
- $A_g \geq 0$ est la **surface** occupée (espace).

Modèle d'un circuit combinatoire

Un circuit est un **graphe orienté acyclique** (DAG) :

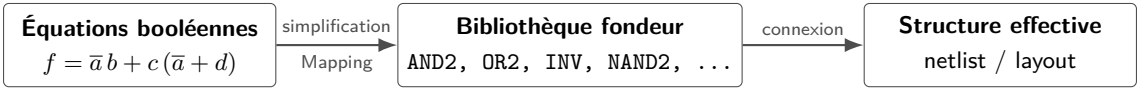
$$G = (V, E)$$

Porte	**Fonction**	A (µm²)	τ (ns)
INV	$\bar{a}$	1	0.05
NAND2	$\overline{ab}$	2	0.08
NOR2	$\overline{a+b}$	2	0.09
AND2	ab	3	0.13
OR2	$a+b$	3	0.14
XOR2	$a \oplus b$	6	0.22

Chiffres indicatifs (nœud technologique générique).

Circuit Combinatoire

Pas de cycle = pas de rebouclage.



Entrées du mapping

- Équations booléennes optimisées
- Bibliothèque de cellules (*std cells*)
- Reformulation selon disponibilité des cellules.
- Contraintes : aire, délai, fanout

Sortie effective

- Réseau de cellules interconnectées
- Respect des *design rules*
- Prêt pour placement/routage

Le mapping traduit une expression logique en une réalisation physique contrainte par la bibliothèque du fondeur.

Circuit : tentative de définition formelle

Définition — Port

Un **port** est un objet typé :

$$p = (\text{nom}, \text{direction}, \text{largeur})$$

avec direction $\in \{in, out\}$ et largeur $\in \mathbb{N}^*$.

Définition — Circuit (combinatoire)

Un **circuit** combinatoire est la donnée : $g = (\text{name}, P_g, f_g, \tau_g, A_g)$ où :

- $P_g = P_g^{in} \uplus P_g^{out}$: ensemble de ports, partitionné en entrées / sorties ;
- $f_g : \mathbb{B}^{|P_g^{in}|} \to \mathbb{B}^{|P_g^{out}|}$: fonction booléenne générale ;
- $\tau_g \in \mathbb{R}_{\geq 0}$: délai caractéristique ;
- $A_g \in \mathbb{R}_{\geq 0}$: surface.

Netlist : Interconnexion de Ports

Définition — Instance de circuit

Une **instance** d'un circuit c **modèle** est un triplet :

$$i = (c, \mathsf{id}, \sigma_i)$$

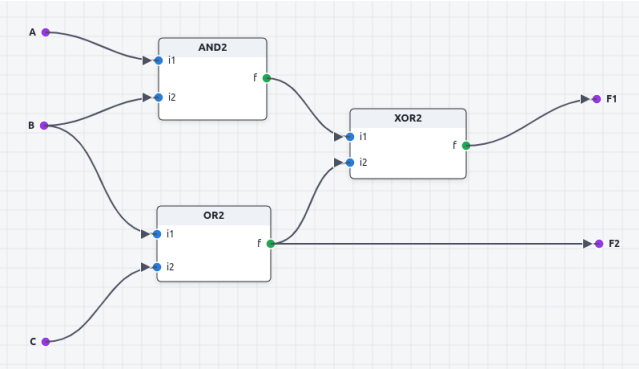
où $\sigma_i : P_c \to N$ associe chaque port de c à un **net** (*fil*, *wire*).
Les ports de l'instance sont notés : $\mathrm{Ports}(i) = \{(i, p) \mid p \in P_c\}$.

Définition — Netlist

Une **netlist** est un quadruplet $\mathcal{N} = (I, N, \Sigma, \mathsf{io})$ où :

- I : ensemble d'instances ;
- N : ensemble de nets ;
- $\Sigma = \{\sigma_i\}_{i \in I}$: connexions port $\to$ net ;
- io : ports externes de l'interface.

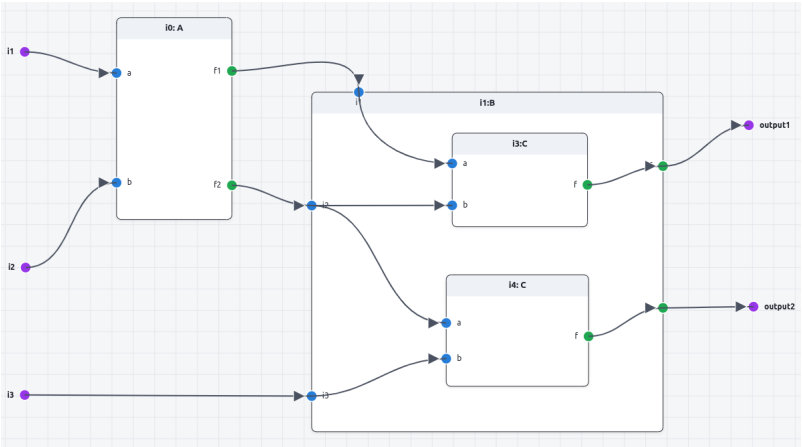
Exemple : netlist élémentaire



Notes :

- partage de ressources (ici `OR2`) entre 2 fonctions $F1$ et $F2$

Exemple : introduction de hiérarchie



Notes :

- `i3` et `i4` sont des *instance* d'un modèle composant (modèle) `C`.

Signaux *pilotés* :

1. **Connexion** : tout net est piloté par un port (*driver*).
2. **Fanout** : un port pilote un ou plusieurs nets.
3. **Risques** : un port piloté par plusieurs drivers conduit à un court-circuit et la destruction du circuit...

Mécanisme d'Encapsulation :

1. Une fois le circuit constitué, on peut le refermer.
2. Sa fonction reste la même, quelle que soient les interactions externes.
3. (Ce n'est pas vrai pour un circuit analogique. Pas de "fonction")

Approche constructive

Un circuit, construit à partir d'instances d'autres circuits, peut lui-même être instancié par la suite, etc.

- L'exemplaire du modèle devient un modèle, etc.
- Une fonction logique peut devenir un composant réutilisable, puis être composée avec d'autres fonctions, sans réintroduire leur physique interne.

De la contrainte physique au contrat comportemental

Analogique : une relation

Un composant impose une contrainte entre les grandeurs de son environnement :

$$F(V, I) = 0$$

Tensions et courants restent *couplés* : le comportement du composant dépend de ce qui lui est connecté.

"lot de contraintes"

"Approche acausale"

Numérique : une fonction

L'encapsulation logique transforme une entrée en une sortie :

$$Y = f(A, B, \ldots)$$

Le détail physique est masqué derrière les niveaux logiques et leurs marges.

"lot de garanties"

"Approche causale"

Chemin Critique : Définition via les Ports

Définition — Chemin

Un **chemin** dans une netlist est une séquence alternée de ports :

$$P = (i_0, p_0) \to (i_1, p_1) \to \cdots \to (i_k, p_k)$$

telle que :

- (i_j, p_j) est un port de **sortie** de g_{i_j} ;
- (i_{j+1}, p_{j+1}) est un port d'**entrée** de $g_{i_{j+1}}$;
- $\sigma_{i_j}(p_j) = \sigma_{i_{j+1}}(p_{j+1})$ (même net).

Délai d'un chemin

$$T(P) \;=\; \sum_{j=0}^{k} \tau_{g_{i_j}}$$

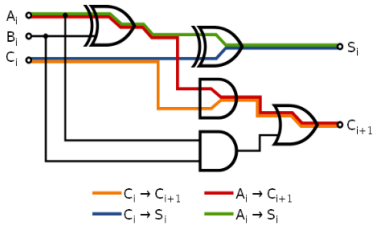
Chemin critique et fréquence

$$T_{\text{crit}} = \max_{P \in \mathcal{P}} T(P), \qquad f_{\max} = \frac{1}{T_{\text{crit}}}$$

où $\mathcal{P}$ parcourt les chemins allant d'un port externe d'entrée à un port externe de sortie.

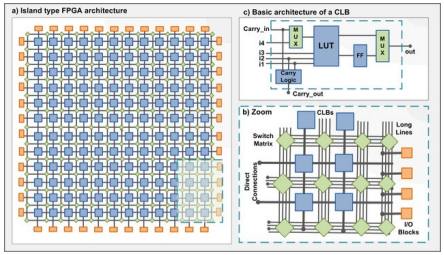
Calcul algorithmique

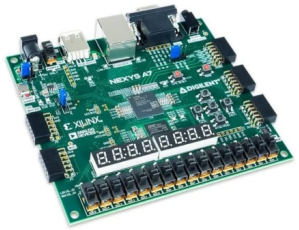
Le calcul se fait en $O(|I| + |\Sigma|)$ par tri topologique du DAG des instances : c'est un **plus long chemin** dans un DAG.



Circuits reconfigurables : FPGA

- Coût élevé d'un circuit **ASIC** *custom*...
- **FPGA** (1985) : circuit *reconfigurable* à volonté.
  - Approche *Software-like*[a]
  - Parallélisme équationnel booléen *massif*.
- Excellents pour :
  - Prototypage, expérimentation.
  - Faible volume de production.
  - Fréquences de fonctionnement modérées.
  - Existent aussi sous forme d'IP.
- Fournisseurs :
  - Xilinx (AMD), Altera (Intel)
  - Lattice, Microchip (Microsemi)
  - *noeuds* technologiques avancés
  - Equivalent de millions de portes logiques.

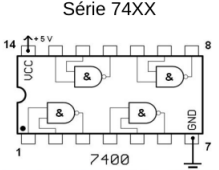


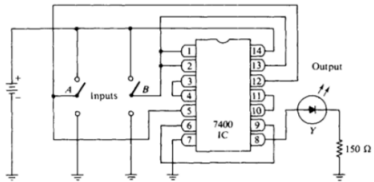


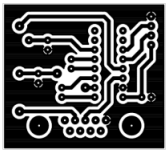
a. voir TD LUT2 où la notion de *bitstream* sera expliquée.

FPGA : loin des circuits discrets !

Série 74XX







Faible capacité en #portes !!!

9– Exemples de circuits combinatoires usuels

Additionneur à propagation de retenue (RCA) : $A + B$

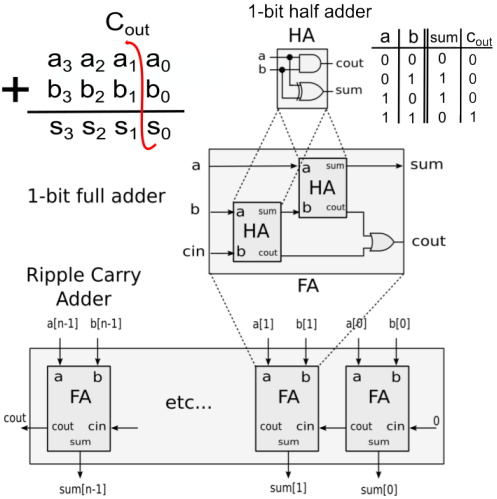
Principe[3] **:** chaîner n additionneurs complets (FA) en propagant la retenue C_i de proche en proche **du LSB vers le MSB**.

Additionneur complet (FA) :

$$S_i = A_i \oplus B_i \oplus C_i$$
$$C_{i+1} = A_i B_i + (A_i \oplus B_i)\, C_i$$

Table de vérité du FA :

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

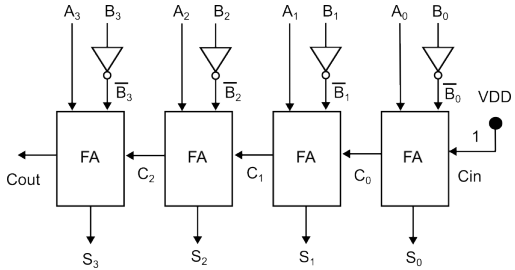


Soustracteur

Principe : réutiliser un additionneur n bits en agissant sur le second opérande via des portes `NOT` et injectant une retenue entrante.

Idée clé : soustraire B revient à ajouter son complément à deux :

$$A - B = A + \overline{B} + 1$$



Point clé

Une seule porte `NOT` par bit + une retenue d'entrée à `1` : l'additionneur devient un *soustracteur* sans modifier sa structure interne.

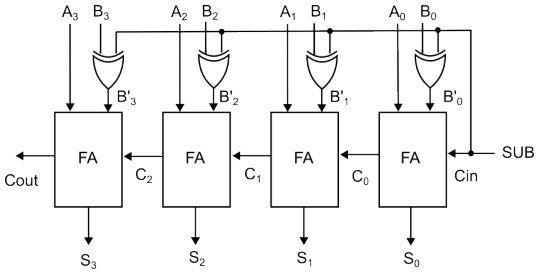
Additionneur-Soustracteur : un composant *programmable*

Principe : réutiliser un additionneur n bits en agissant sur le second opérande via des portes `XOR` commandées par un signal sub également utilisé en retenue *entrante*.

Idée clé : soustraire B revient à ajouter son complément à deux :

$$A - B = A + \overline{B} + 1$$

- **sub = 0** : $B' = B \oplus 0 = B$, $C_{in} = 0 \Rightarrow$ **addition** $A + B$
- **sub = 1** : $B' = B \oplus 1 = \overline{B}$, $C_{in} = 1 \Rightarrow$ **soustraction** $A - B$



Point clé

Une seule porte XOR par bit + une retenue d'entrée : l'additionneur devient un *additionneur-soustracteur* sans modifier sa structure interne.

Multiplieur (Mult)

Principe : suite d'additions décalées (algorithme « papier-crayon »).

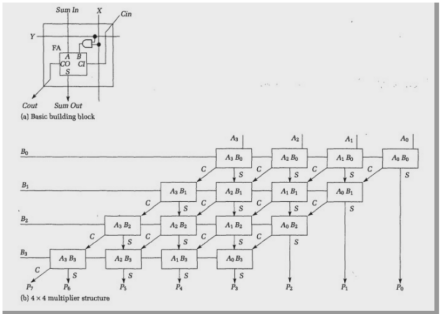
Formule booléenne (produit partiel) :

$$PP_j = A \times b_j = A \cdot b_j \quad \text{(ET bit à bit)}$$

Résultat :

$$P = \sum_{j=0}^{n-1} \left(A \times b_j\right) \ll j$$

	a_3	a_2	a_1	a_0
$\times$	b_3	b_2	b_1	b_0
	a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_3b_1	a_2b_1	a_1b_1	a_0b_1	
a_2b_2	a_1b_2	a_0b_2		
a_1b_3	a_0b_3			
p_4	p_3	p_2	p_1	p_0



Multiplexeur 2 vers 1 (MUX21)

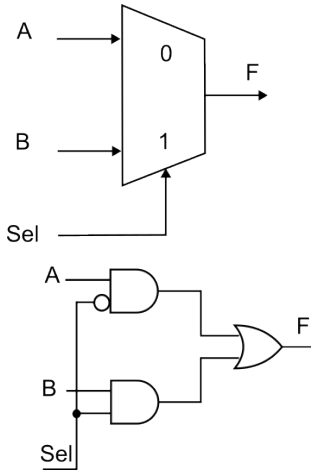
Rôle : sélectionner une entrée parmi deux selon un signal de commande.

Table de vérité :

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

Formule booléenne :

$$F = \overline{Sel} \times A + Sel \times B$$



Shifter à position fixe (décalage par câblage)

Idée clé : décaler une donnée d'un nombre **constant** de positions ne nécessite **aucune porte logique**– un simple **recâblage** suffit.

Formule booléenne (décalage logique à gauche de k) :

$$S_i = \begin{cases} A_{i-k} & \text{si } i \geq k \\ 0 & \text{si } i < k \end{cases}$$

Décalage logique à droite de k :

$$S_i = \begin{cases} A_{i+k} & \text{si } i + k < n \\ 0 & \text{sinon} \end{cases}$$

Chaque sortie S_i est reliée *directement* à une entrée A_j ou à la masse (0).

Exemple : A sur 4 bits, SLL de 1 ($k = 1$)

	S_3	S_2	S_1	S_0
Connexion	A_2	A_1	A_0	0

Aucun composant actif : juste des fils.

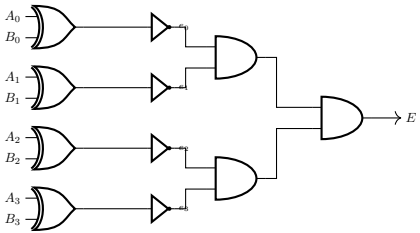
Comparateur d'égalité de 2 mots binaires

Objectif : produire un signal $E = 1$ si et seulement si $A = B$, avec $A = A_{n-1} \dots A_0$ et $B = B_{n-1} \dots B_0$.

Principe : comparer *bit à bit* puis combiner.

$$e_i = \overline{A_i \oplus B_i} \qquad \text{(XNOR)}$$

$$E = \prod_{i=0}^{n-1} \overline{A_i \oplus B_i}$$



- $A_i = B_i \Rightarrow e_i = 1$
- $A_i \neq B_i \Rightarrow e_i = 0$ (et $E = 0$)

Point clé

n portes **XNOR** en parallèle (une par bit) + un **ET** global (arbre d'AND) pour agréger les égalités partielles.

Décodeur $n \to 2^n$

Principe : Un décodeur $n \to 2^n$ active une seule sortie parmi 2^n en fonction du code binaire présent en entrée.

Entrées : A_1, A_0 **Sorties :** S_0, S_1, S_2, S_3

Équations booléennes :

$$S_0 = \overline{A_1}\,\overline{A_0}$$
$$S_1 = \overline{A_1}\,A_0$$
$$S_2 = A_1\,\overline{A_0}$$
$$S_3 = A_1\,A_0$$

Forme compacte :

$$S_i = \prod_{k=0}^{n-1} \left(A_k \oplus \overline{b_{i,k}}\right)$$

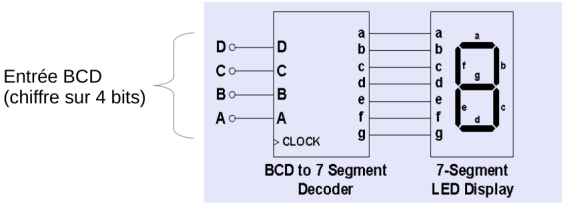
où $b_{i,k}$ est le k-ième bit de la représentation binaire de l'indice i.

Tableau de principe :

A_1	A_0	S_0	S_1	S_2	S_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**

Exemple : pour $i = 2$, on a $(b_{2,1}\, b_{2,0})_2 = (10)_2$, donc $b_{2,1} = 1$ et $b_{2,0} = 0$.

Décodeur BCD vers afficheur 7 segments



Binary Inputs				Decoder Outputs							7-Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

7 fonctions binaires (a...g) des entrées (A,B,C,D)