

Chapitre 01 : Logique Booléenne

"Au commencement était le NAND."



Où en sommes-nous ?

Stack des couches d'abstraction

Les 8 couches d'abstraction — nous sommes à la couche 1

Nous posons les **fondations** de tout l'édifice !

Pourquoi le Binaire ?

Question : Pourquoi 0 et 1, pas 0-9 ?

Réponses :

1. **Fiabilité** : Distinguer 2 états est plus robuste que 10
2. **Simplicité** : Un transistor = un interrupteur (on/off)
3. **Universalité** : George Boole (1854) : toute logique = Vrai/Faux



Concept Clé

Moins d'états = plus de tolérance au bruit électrique

Du Voltage au Bit

Tension	Signification
0V - 0.8V	0 (Faux)
2.4V - 3.3V	1 (Vrai)

La zone 0.8V-2.4V est **interdite** — c'est cette séparation nette qui rend le binaire robuste.



On retrouve '0' et '1' comme valeurs de type `std_logic`.

La Porte NAND : Notre Axiome

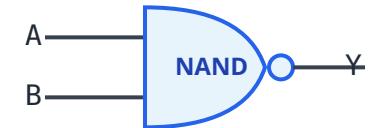
Pourquoi partir du NAND ?

1. Complétude fonctionnelle :

TOUTES les portes peuvent être construites à partir de NAND

2. Réalité physique : En CMOS,
NAND = seulement 4 transistors

3. Pédagogie : Une seule brique → comprendre l'abstraction



Symbole de la porte NAND

Table de Vérité NAND

A	B	NAND(A, B)
0	0	1
0	1	1
1	0	1
1	1	0

Règle NAND

Le résultat est 0 *seulement si A et B sont à 1.*

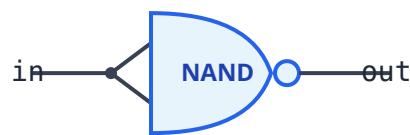
NAND = "Not-AND" = inverse d'un AND

Universalité du NAND

```
flowchart LR
    NAND[NAND] --> NOT[NOT]
    NAND --> AND[AND]
    NAND --> OR[OR]
    NOT --> XOR[XOR]
    AND --> XOR
    OR --> XOR
    AND --> MUX[MUX]
    OR --> MUX
    NOT --> MUX
    NOT --> DMUX[DMUX]
    AND --> DMUX
```

Construction : NOT (Inverseur)

Astuce : Connecter le même signal aux deux entrées !



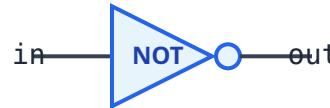
$$\text{NOT} = \text{NAND}(a, a)$$

Vérification :

- Si $\text{in} = 0$: $\text{NAND}(0, 0) = 1 \checkmark$
- Si $\text{in} = 1$: $\text{NAND}(1, 1) = 0 \checkmark$

in	NOT(in)
0	1
1	0

Symbole de la porte NOT



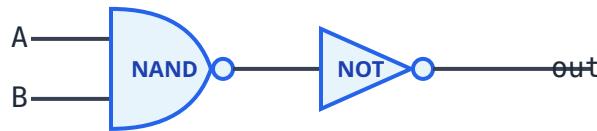
Symbole standard de l'inverseur



En VHDL : `y <= not a;`

Construction : AND

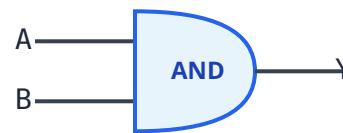
Insight : $\text{NOT}(\text{NAND}(A, B)) = \text{AND}(A, B)$



$$\text{AND} = \text{NOT}(\text{NAND}(a, b))$$

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Symbole de la porte AND



Symbole standard de la porte AND

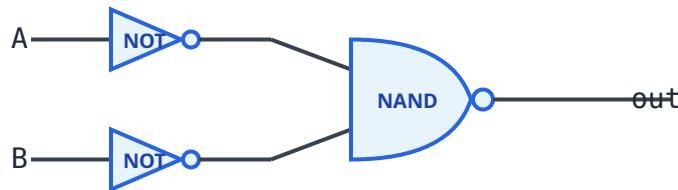


En VHDL : `y <= a and b;`

Construction : OR

Théorème de De Morgan :

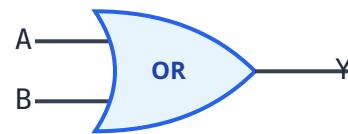
$$\begin{aligned} A \text{ OR } B &= \text{NOT((NOT A) AND (NOT B))} \\ &= (\text{NOT A}) \text{ NAND } (\text{NOT B}) \end{aligned}$$



OR avec 3 portes NAND

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

Symbole de la porte OR



Symbole standard de la porte OR



En VHDL : `y <= a or b;`

Construction : XOR (Ou Exclusif)

Rôles cruciaux du XOR :

- Addition binaire (somme sans retenue)
- Comparaison (bits différents ?)
- Cryptage (chiffrement par flux)
- Calcul de parité

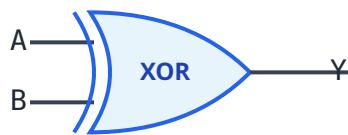
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Règle XOR

Sortie = 1 si et seulement si les entrées sont **définitivement différentes**

XOR : Détail de Construction

Formule : $A \text{ XOR } B = (A \text{ AND } \text{NOT } B) \text{ OR } (\text{NOT } A \text{ AND } B)$



Symbole de la porte XOR

Autre formule équivalente :

$A \text{ XOR } B = (A \text{ OR } B) \text{ AND } \text{NOT}(A \text{ AND } B)$



VHDL

En VHDL : `y <= a xor b;`

XOR : Propriétés Utiles

Propriété	Expression	Usage
Identité	$A \text{ XOR } 0 = A$	Masquage sélectif
Inversion	$A \text{ XOR } 1 = \text{NOT } A$	Inversion conditionnelle
Auto-inverse	$A \text{ XOR } A = 0$	Mise à zéro rapide
Commutativité	$A \text{ XOR } B = B \text{ XOR } A$	Simplification
Associativité	$(A \text{ XOR } B) \text{ XOR } C = A \text{ XOR } (B \text{ XOR } C)$	Parité



En ARM, l'instruction EOR (Exclusive OR) utilise ces propriétés : EOR R0, R0, R0 met R0 à zéro en 1 cycle.

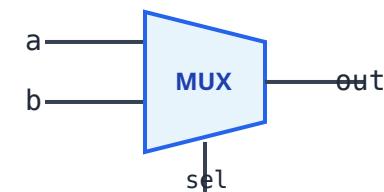
Le Multiplexeur (Mux) — L'Aiguilleur

Le composant le plus important !

- Si `sel == 0` → `out = a`
- Si `sel == 1` → `out = b`

Formule :

$$\text{out} = (\text{a AND NOT sel}) \text{ OR } (\text{b AND sel})$$



Mux 2:1

Pourquoi le Mux est Crucial ?

Dans un CPU, à chaque cycle il faut choisir :

- D'où vient l'opérande ?
 - Mémoire ou registre ?
- Où va le résultat ?
 - Mémoire ou registre ?
- Quelle instruction ?
 - ADD, SUB, AND... ?

🔗 ARM

Le CPU ARM utilise des Mux pour sélectionner parmi les registres R0-R15. Le champ de registre (4 bits) commande un Mux 16:1.

Chaque choix = un Mux !

Mux Étendu : Mux4Way, Mux8Way

Pour sélectionner parmi N entrées, il faut $\log_2(N)$ bits de sélection :

Mux	Entrées	Bits sel
Mux2	2	1
Mux4	4	2
Mux8	8	3
Mux16	16	4

Construction hiérarchique

Un Mux4Way = 3 Mux2Way en arbre
Logique Booléenne

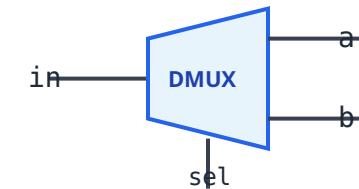
Le Démultiplexeur (DMux)

L'inverse du Mux : 1 entrée → N sorties

- Si `sel == 0` → `a = in, b = 0`
- Si `sel == 1` → `a = 0, b = in`

Usages :

- Adressage mémoire
- Routage des signaux
- Décodage d'instructions



DMux 1:2

Portes Multi-Bits (Bus)

Pour traiter des mots de 32 bits en parallèle :

```
entity And32 is
  port(
    a : in bits(31 downto 0);
    b : in bits(31 downto 0);
    y : out bits(31 downto 0)
  );
end entity;
```

```
architecture rtl of And32 is
begin
  gen: for i in 0 to 31 generate
    u: And2 port map (
      a => a(i),
      b => b(i),
      y => y(i)
    );
  end generate;
end architecture;
```

Notion de Bus

Bus

Groupe de fils transportant des données multi-bits en parallèle. Notation :
bits(31 downto 0) = 32 fils.

Avantages :

- Transfert parallèle (32 bits/cycle)
- Notation compacte
- Opérations vectorielles

Exemples de bus :

- Bus de données : 32 bits
- Bus d'adresse : 32 bits
- Bus de contrôle : 8-16 bits

Portes Multi-Entrées

Or8Way : Teste si au moins 1 bit parmi 8 est à 1

```
Or8Way(a[0..7]) = a[0] OR a[1] OR ... OR a[7]
```

Construction en arbre (3 niveaux au lieu de 7) :

Niveau 1: Or2(a[0],a[1])→t0 Or2(a[2],a[3])→t1 ...

Niveau 2: Or2(t0, t1) → t4 Or2(t2, t3) → t5

Niveau 3: Or2(t4, t5) → sortie

Pourquoi l'arbre ?

Profondeur $\log_2(N)$ au lieu de $N-1 \rightarrow$ moins de délai de propagation
Logique Booléenne

HDL : Description Matérielle

```
entity And2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of And2 is
  component Nand port(a,b: in bit; y: out bit); end component;
  component Inv port(a: in bit; y: out bit); end component;
  signal w : bit;
begin
  u1: Nand port map (a => a, b => b, y => w);
  u2: Inv port map (a => w, y => y);
end architecture;
```

Pont avec VHDL Professionnel



nand2c HDL	VHDL standard
bit	std_logic
bits(31 downto 0)	std_logic_vector(31 downto 0)
port map	Identique !
for generate	Identique !

Vous apprenez la vraie syntaxe VHDL !

Exemple Tracé : Calcul AND

Traçons $\text{AND}(1, 0)$ pas à pas :

1 Entrées

$$A = 1, B = 0$$

2 NAND(A, B)

$$\text{NAND}(1, 0) = 1$$

3 NOT du résultat

$$\text{NOT}(1) = 0 \rightarrow \text{AND}(1, 0) = 0 \checkmark$$

Exemple Tracé : Calcul Mux

Traçons $\text{Mux}(a=1, b=0, \text{sel}=1)$:

- 1 $\text{NOT}(\text{sel}) = \text{NOT}(1) = 0$
- 2 $a \text{ AND NOT(sel)} = 1 \text{ AND } 0 = 0$
- 3 $b \text{ AND sel} = 0 \text{ AND } 1 = 0$
- 4 $0 \text{ OR } 0 = 0$

Résultat : $\text{out} = b = 0$ quand $\text{sel} = 1$ ✓

Du NAND au CPU : La Feuille de Route

Roadmap NAND vers CPU

Progression à travers les chapitres

Rôle de Chaque Porte dans le CPU

Porte	Rôle dans le CPU
NOT	Soustraction (complément à 2), inversion de condition
AND	Masquage de bits, conditions ET, extraction de champs
OR	Combinaison de signaux, conditions OU
XOR	Addition bit à bit, comparaison, parité
Mux	Tous les choix du datapath
DMux	Adressage mémoire, décodage instruction

Questions de Réflexion

1. Pourquoi le NAND est-il préféré au NOR en CMOS ?
2. Combien de portes NAND faut-il pour un XOR ?
3. Quel est l'avantage d'un Mux large (16:1) vs plusieurs petits ?
4. Comment faire un Mux 4:1 avec des Mux 2:1 ?
5. Quelle est la profondeur d'un Or16Way en arbre ?

Synthèse des Portes

A	B	NAND	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

Mnémotechnique

AND = tous à 1 | **OR** = au moins un à 1 | **XOR** = nombre impair de 1

Ce qu'il faut retenir

1. Le binaire simplifie : 2 états plus fiables que 10
2. NAND est universel : Toutes les portes en découlent
3. L'abstraction est puissante : Couches les unes sur les autres
4. Mux = choix, DMux = routage
5. XOR = addition, comparaison, parité
6. Bus = traitement parallèle de plusieurs bits

Questions ?



Référence : Livre Seed, Chapitre 01 - Logique Booléenne

👉 **Exercices :** TD et TP disponibles

Prochain chapitre : Arithmétique (ALU)