

Ch. 4

Éléments de Conception Logique

B. Quoitin
(bruno.quoitin@umons.ac.be)

Table des Matières

Introduction

Objectifs

- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

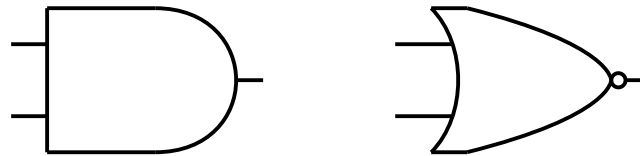
Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Objectifs

Des portes logiques aux blocs de base d'un processeur

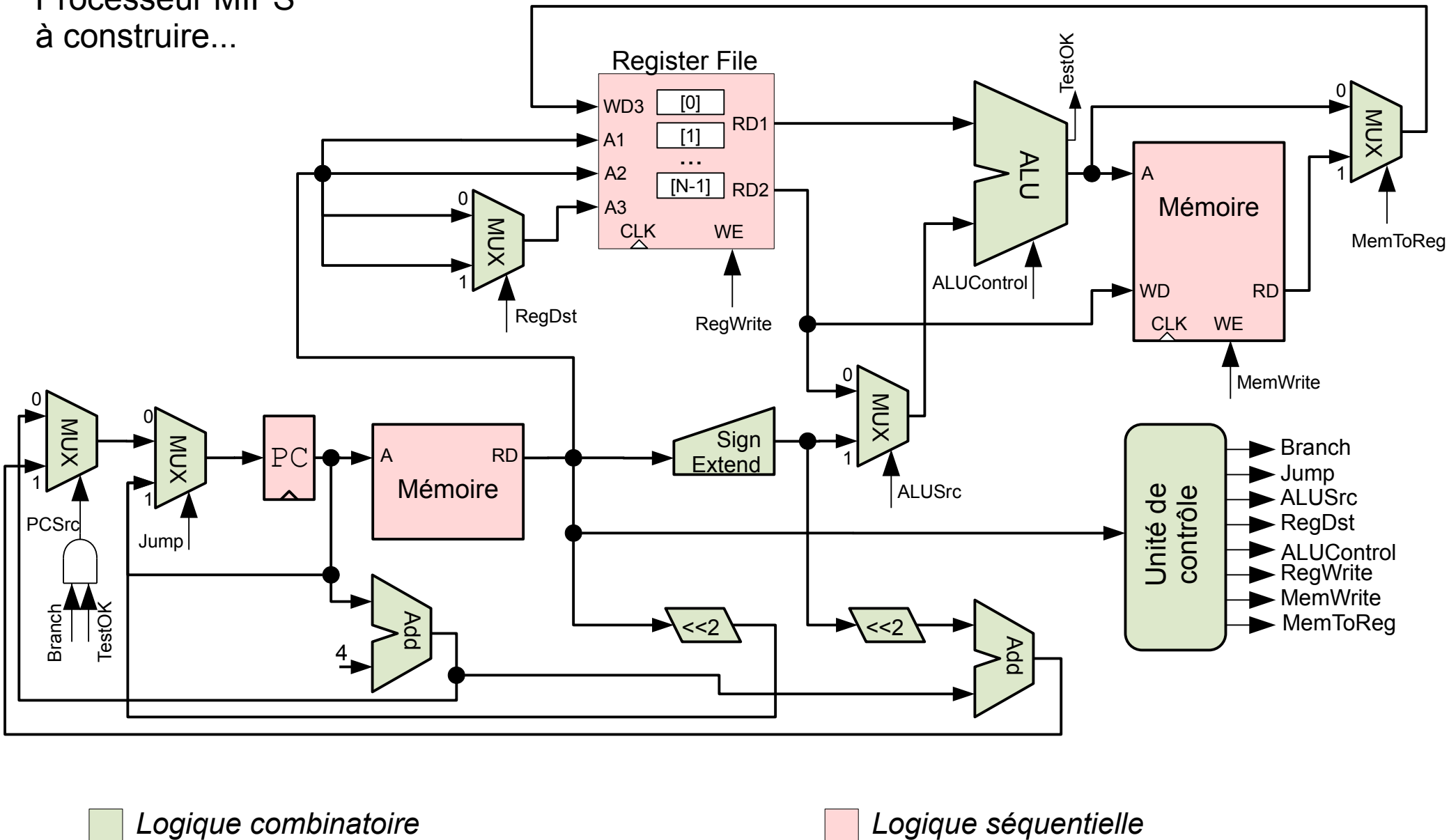
- Comprendre comment les opérations arithmétiques de base et la logique de contrôle d'un processeur peuvent être réalisées à partir d'un assemblage complexe d'opérations logiques élémentaires : les **portes logiques**.



- Introduire un formalisme général permettant de raisonner sur un assemblage de portes logiques : **algèbre de commutation**.
- Donner l'intuition de la correspondance entre le monde logique / digital et le monde physique à l'aide de circuits électriques simples.
- Introduire un certain nombre de **blocs de base** qui seront utiles pour construire un processeur par la suite.

Objectifs

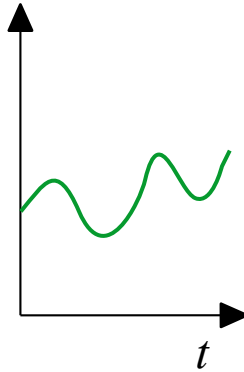
Processeur MIPS
à construire...



Introduction

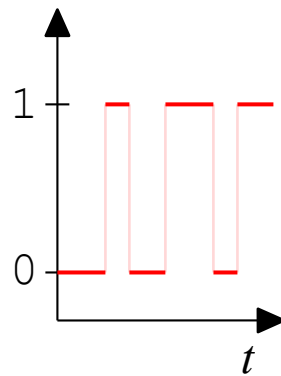
Monde analogique

- Les systèmes analogiques sont ceux qui traitent des signaux qui lorsqu'ils varient dans le temps peuvent prendre une **infinité de valeurs différentes** dans un intervalle donné.
- Par exemple: une tension ou un courant électriques, une température, une pression, ...



Monde digital

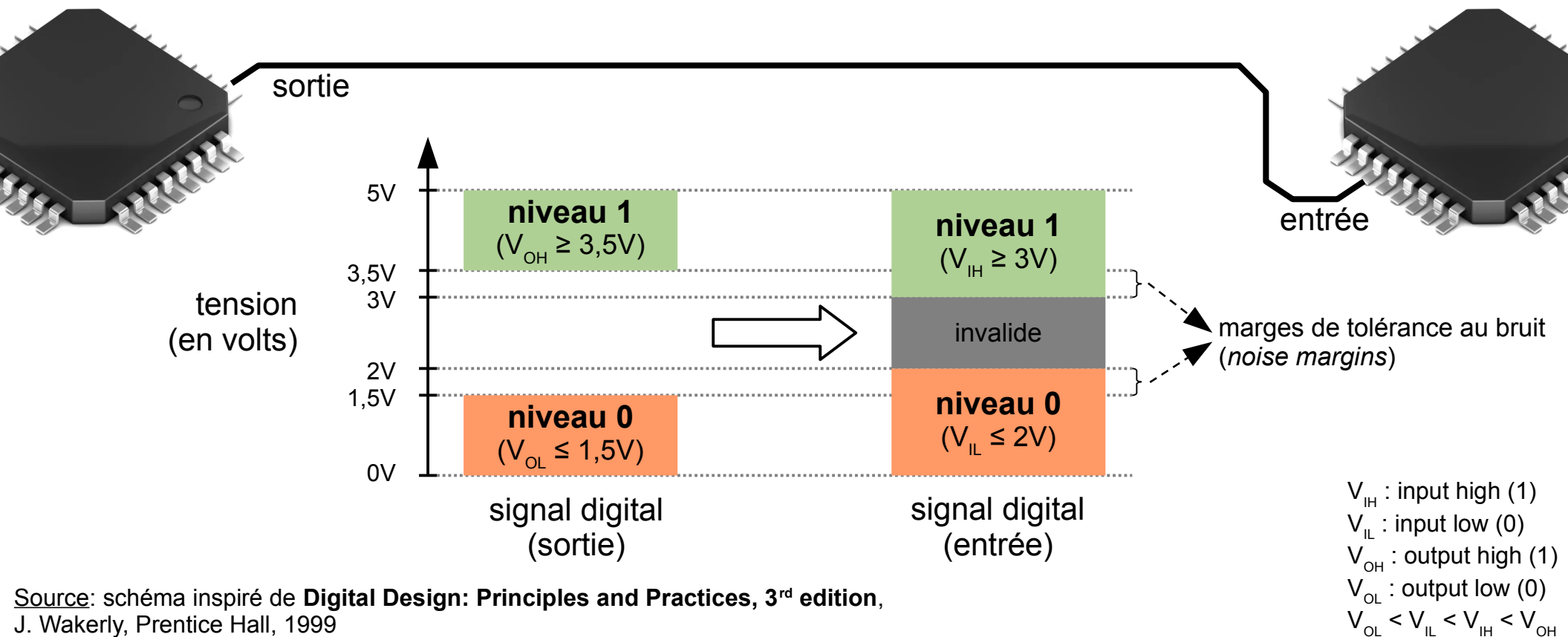
- Dans un monde digital, les signaux peuvent également prendre un nombre infini de valeurs, mais ils sont classés de façon à conserver un **nombre fini de valeurs différentes** (typiquement 2).
- Un **signal logique** ou **signal digital** prend des valeurs désignées conventionnellement par 0/1 (ou off/on, faux/vrai, bas/haut, ...)



Introduction

Correspondance analogique / digital

- Un signal digital est souvent implémenté en segmentant les valeurs prises par un signal analogique en plusieurs intervalles.
- Exemple : **sortie** d'un système digital et **entrée** d'un autre ; signal digital implémenté par signal analogique (tension électrique). Niveaux logiques (0 et 1) associés à des intervalles de tension électrique différents (convention).



Source: schéma inspiré de **Digital Design: Principles and Practices**, 3rd edition, J. Wakerly, Prentice Hall, 1999

Introduction

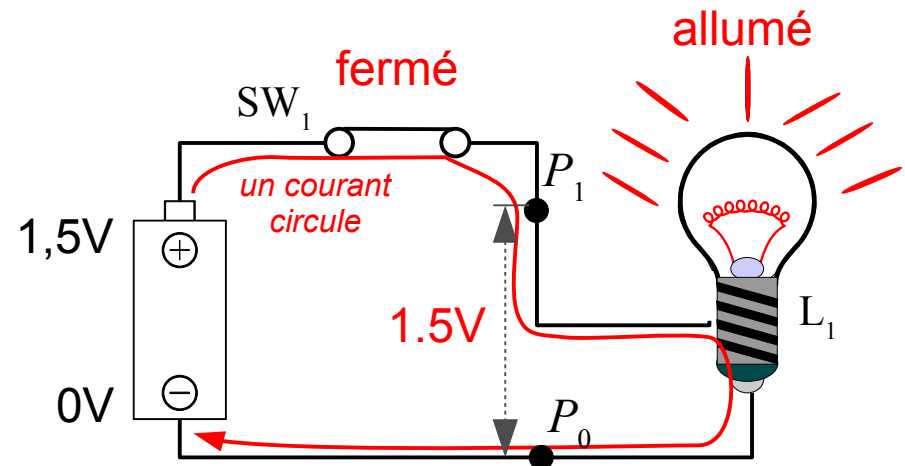
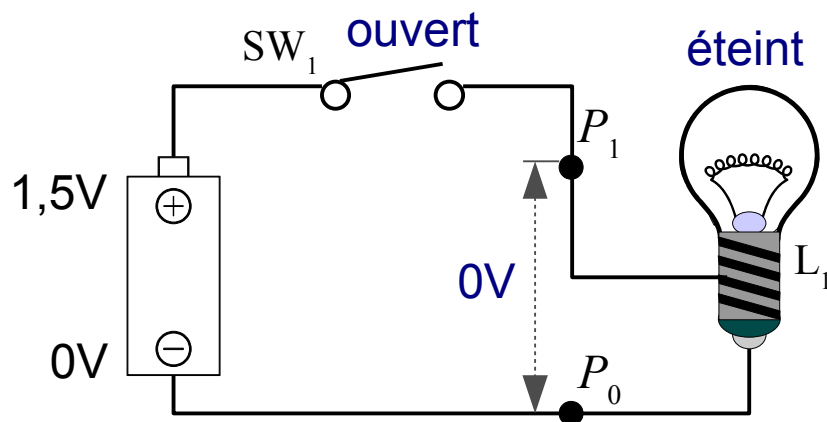
Avantages du monde digital

- Reproductibilité des résultats
 - pour des signaux en entrée donnés, un système digital produit toujours exactement le même signal digital en sortie.
- Facilité de conception
 - un circuit digital simple peut être compris sans connaître les détails électroniques ou physiques de l'implémentation.
- Programmable
 - les circuits digitaux peuvent être décrits en utilisant des langages de programmation spécifiques: *Hardware Design Languages* (HDL). Par exemple: VHDL ou Verilog.
- Vitesse
 - les transistors utilisés pour implémenter les systèmes digitaux contemporains ont des **temps de commutation** très faibles, de l'ordre de quelques picosecondes (10^{-12} secondes).

Introduction

Circuits électriques digitaux

- Afin de donner une **intuition** du fonctionnement des circuits digitaux, nous nous baserons sur de simples circuits électriques composés de piles, ampoules et interrupteurs.



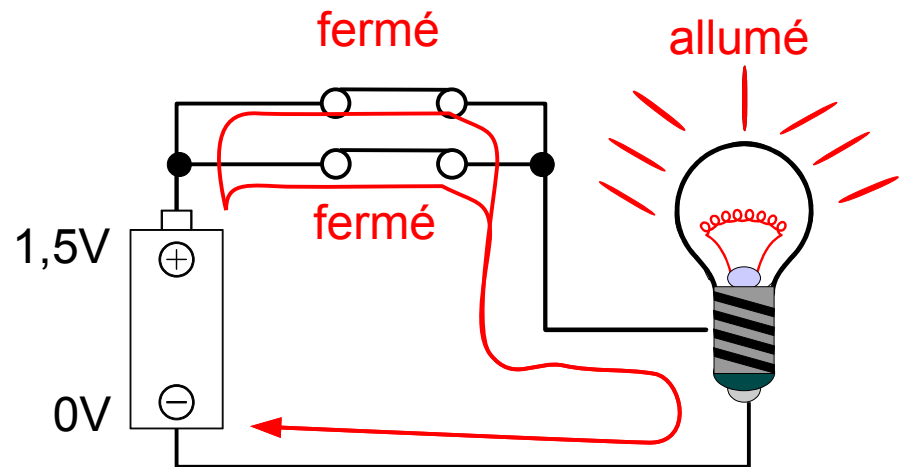
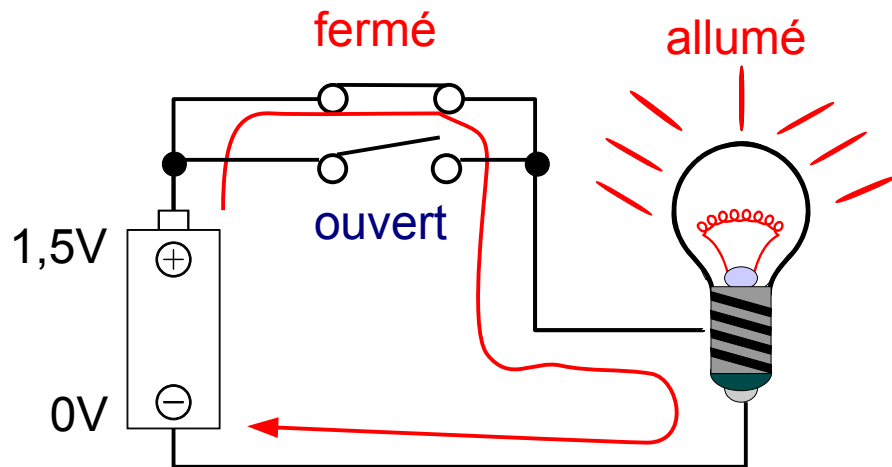
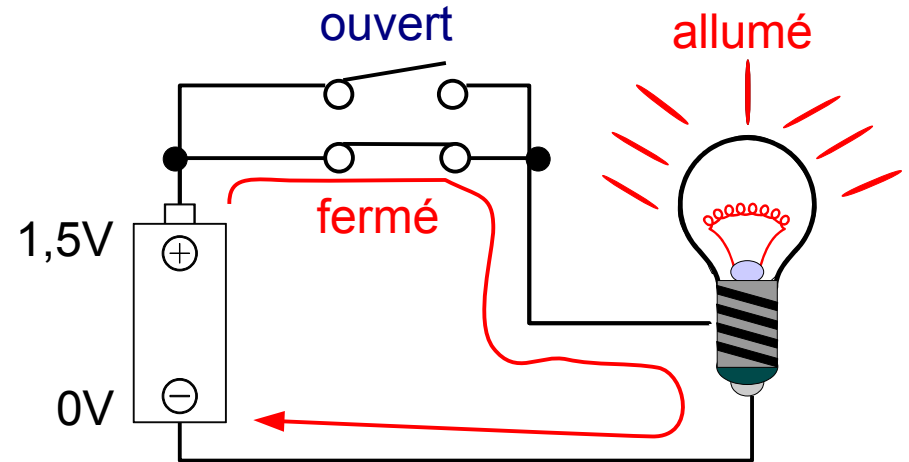
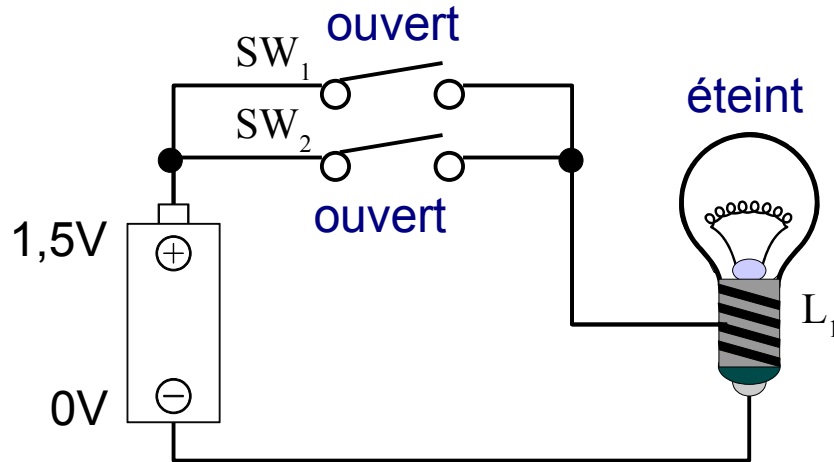
- Quels sont les états mesurables dans le circuit ci-dessus ?
 - interrupteur SW1 : **ouvert** / **fermé**
 - différence de tension entre P₁ et P₀ : **0V** / **1,5V**
 - ampoule L1 : **éteinte** / **allumée**

$$L_1 = SW_1$$

Introduction

Opérateurs logique - OU

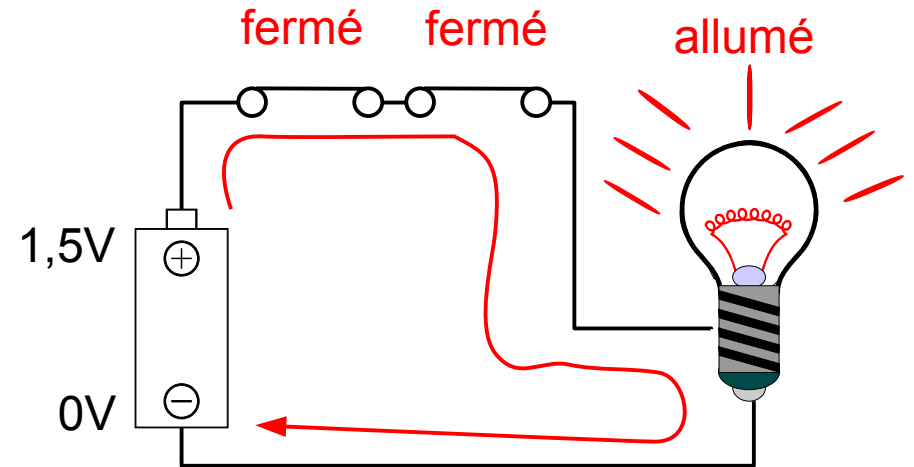
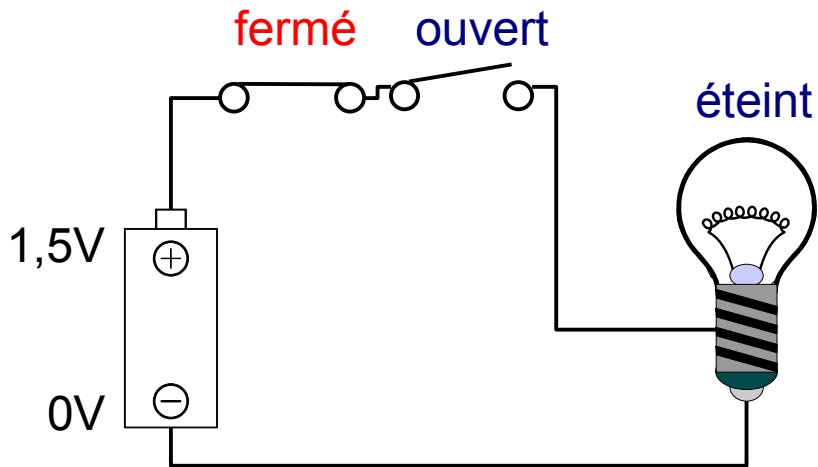
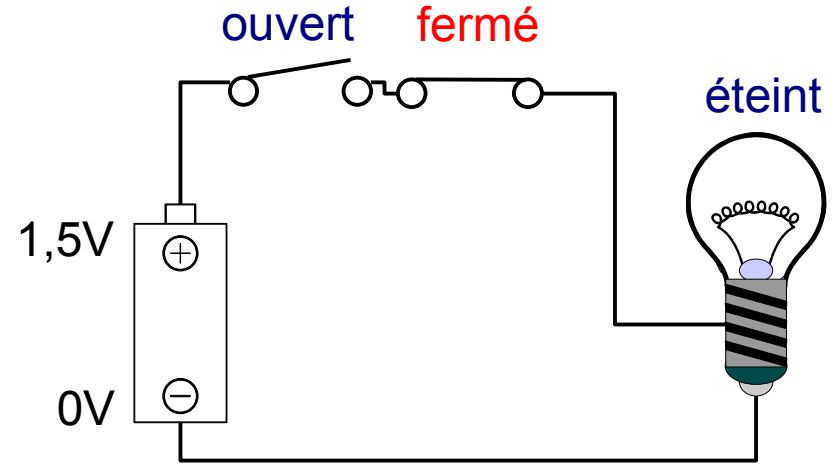
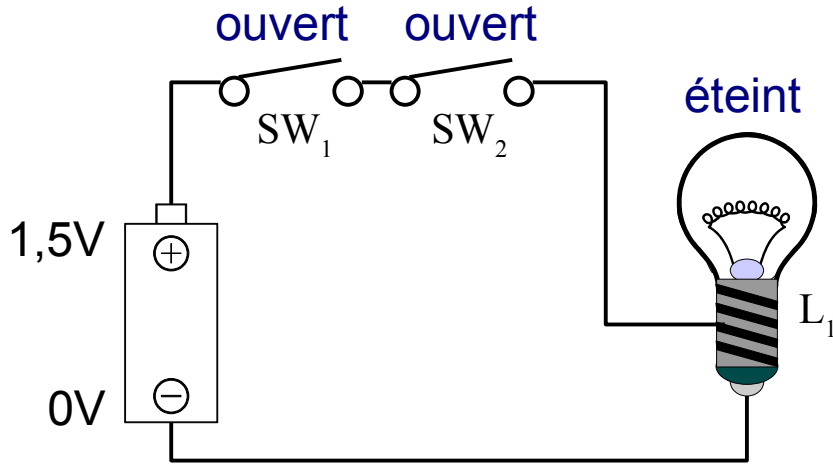
$$L_1 = SW_1 \text{ OU } SW_2$$



Introduction

Opérateurs logique - ET

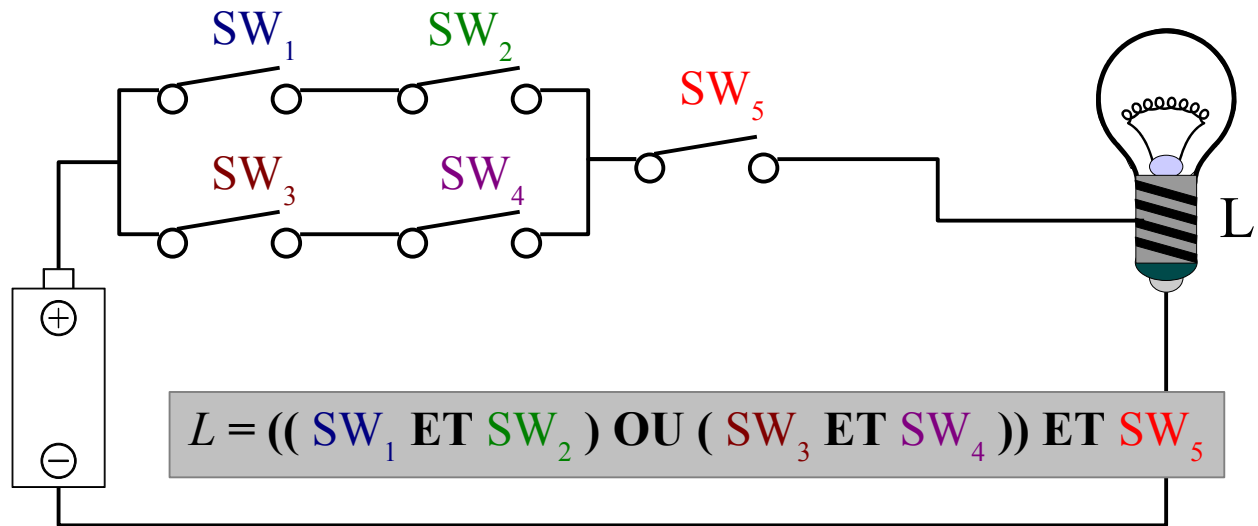
$$L_1 = SW_1 \text{ ET } SW_2$$



Introduction

Combinaison complexe d'opérateurs

- Quelle est la fonction logique implémentée par cette combinaison d'interrupteurs ?



- **Conclusion** : Nécessité de pouvoir raisonner sur des systèmes digitaux plus complexes.

Table des Matières

Introduction

- Objectifs
- Circuits logiques

Algèbre de Commutation

- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Algèbre de Commutation

Algèbre de Boole

- Un ensemble E a une structure d'**algèbre de Boole** s'il est muni de lois de composition internes, **addition** (notée « + ») et **multiplication** (notée « . ») qui satisfont les propriétés suivantes :
 - elles sont **commutatives**
 - elles sont **associatives**
 - elles sont **distributives** l'une par rapport à l'autre
 - elles admettent un élément **neutre** (noté 0 pour l'addition et 1 pour la multiplication)
 - elles sont telles que tout élément de E est **idempotent** pour ces lois, i.e. $x + x = x$ et $x . x = x$
- et tel que chaque élément x possède un unique **complément** x' vérifiant
 - $x + x' = 1$ (*loi du tiers exclu*)
 - $x . x' = 0$ (*principe de contradiction*)

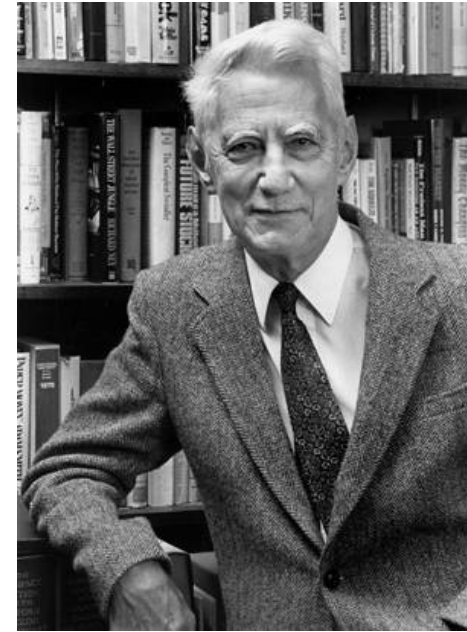


Boole

Algèbre de Commutation

Algèbre de commutation

“In [...] circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. [...] In this paper⁽¹⁾ a mathematical analysis of certain of the properties of such networks will be made.”



Shannon

- Une **algèbre de commutation** est une algèbre de Boole à 2 éléments définie sur l'ensemble $B=\{0, 1\}$ et munie de lois de composition internes et d'un complément définis par les **postulats de base** suivants

Addition logique / disjonction / **OU**

$$1 + 1 = 1 = 1 + 0 = 0 + 1 \quad (\text{P1})$$

$$0 + 0 = 0 \quad (\text{P2})$$

Complément / **NON**

$$0' = 1 \quad (\text{P3})$$

$$1' = 0 \quad (\text{P3}^*)$$

Multiplication logique / conjonction / **ET**

$$1.1 = 1 \quad (\text{P1}^*)$$

$$0.0 = 0 = 0.1 = 1.0 \quad (\text{P2}^*)$$

(1) Shannon, C. E., **A Symbolic Analysis of Relay and Switching Circuits**, American Institute of Electrical Engineers Transactions, 57 (1938).

Algèbre de Commutation

Théorèmes à une variable

- Neutre

- pour l'addition (T1)
- pour la multiplication (T1*)

$$\forall x \in B, x + 0 = x$$

$$\forall x \in B, x \cdot 1 = x$$

- Absorbant

- pour l'addition (T2)
- pour la multiplication (T2*)

$$\forall x \in B, x + 1 = 1$$

$$\forall x \in B, x \cdot 0 = 0$$

- Idempotence

- pour l'addition (T3)
- pour la multiplication (T3*)

$$\forall x \in B, x + x = x$$

$$\forall x \in B, x \cdot x = x$$

- Involution (T4)

$$\forall x \in B, (x')' = x$$

- Principe du “tiers exclu” (T5)

$$\forall x \in B, x + x' = 1$$

- Principe de contradiction (T5*)

$$\forall x \in B, x \cdot x' = 0$$

Algèbre de Commutation

Théorèmes à plusieurs variables

- Commutativité

- de l'addition (T6)

$$\forall x, y \in B, x + y = y + x$$

- de la multiplication (T6*)

$$\forall x, y \in B, x \cdot y = y \cdot x$$

- Associativité

- de l'addition (T7)

$$\forall x, y, z \in B, (x + y) + z = x + (y + z)$$

- de la multiplication (T7*)

$$\forall x, y, z \in B, (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

- Distributivité

- de la multiplication sur l'addition (T8)

$$\forall x, y, z \in B, x \cdot (y + z) = x \cdot y + x \cdot z$$

- de l'addition sur la multiplication (T8*)

$$\forall x, y, z \in B, x + (y \cdot z) = (x + y) \cdot (x + z)$$

il est intéressant de remarquer que T8* n'est pas vrai dans l'algèbre que nous utilisons couramment

Algèbre de Commutation

Théorèmes à plusieurs variables

- Couverture ou absorption

- (T9) $\forall x, y \in B, x + x \cdot y = x$

“x couvre x.y”

- (T9*) $\forall x, y \in B, x \cdot (x + y) = x$

- Combinaison

- (T10) $\forall x, y \in B, x \cdot y + x \cdot y' = x$

- (T10*) $\forall x, y \in B, (x + y) \cdot (x + y') = x$

- Consensus

- (T11) $\forall x, y, z \in B, (x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$

- (T11*) $\forall x, y, z \in B, x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$

- De Morgan

- (T13) $\forall x, y \in B, (x \cdot y)' = x' + y'$

- (T13*) $\forall x, y \in B, (x + y)' = x' \cdot y'$

Note: T9, T9*, T10 et T10* sont à la base de techniques de minimisation telles que les tableaux de Karnaugh et l'algorithme de Quine-McCluskey.

Algèbre de Commutation

Table de vérité

- Une **table de vérité** est un moyen d'exprimer une fonction logique *en extension*. Pour chaque combinaison des entrées de la fonction, le résultat de la fonction est donné dans la table.
- Ainsi, une fonction à N arguments est représentée par une table de vérité à 2^N lignes et à $N+1$ colonnes.

- Exemple :
fonction de 3 variables
(addition modulo 2).

N variables				2^N lignes
	x	y	z	
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	
$F(x,y,z)$				

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation

→ Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

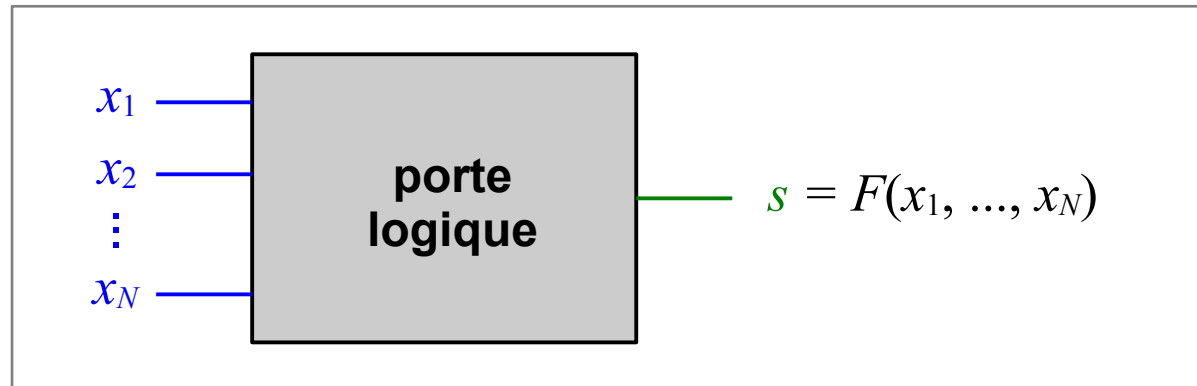
Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Portes Logiques

Définition

- Une **porte logique** (*logic gate*) est une implémentation, en général électronique, d'une fonction logique $F(x_1, \dots, x_N) : B^N \rightarrow B$.
- Une porte logique est représentée par un **symbole** indiquant ses entrées (arguments) d'une part et ses sorties (résultats) d'autre part.
 - x_1 à x_N sont les **signaux logiques présentés en entrée**
 - s est le **signal logique émis en sortie**



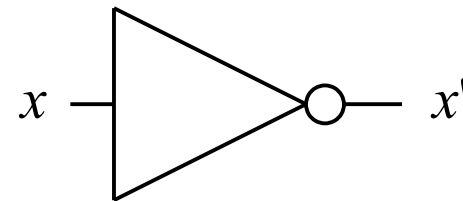
- Les signaux présentés sur les ports d'entrée et émis sur les ports de sorties sont typiquement des tensions électriques qui représentent des états logiques 0 et 1.

Portes Logiques

Portes logiques communes

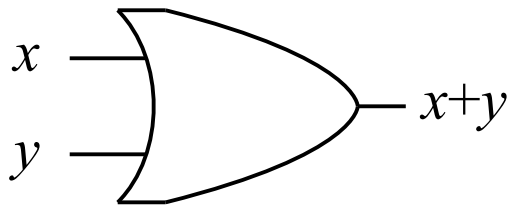
- Les portes logiques les plus communes sont désignées par des symboles standard.

NON (NOT)



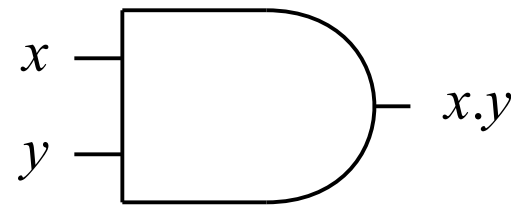
x	x'
0	1
1	0

OU (OR)



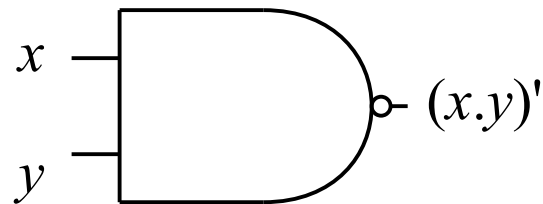
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

ET (AND)



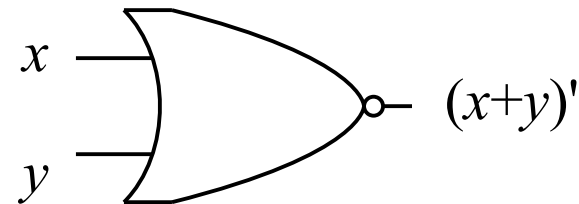
x	y	$x.y$
0	0	0
0	1	0
1	0	0
1	1	1

NON-ET (NAND)



x	y	$(x.y)'$
0	0	1
0	1	1
1	0	1
1	1	0

NON-OU (NOR)



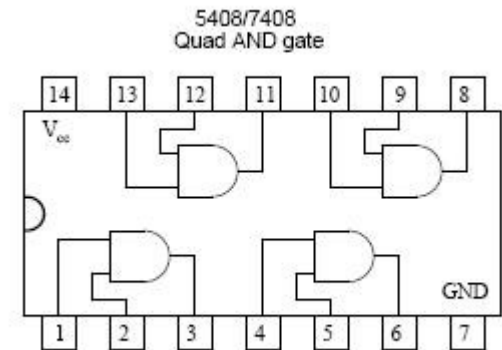
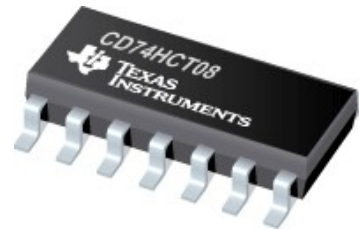
x	y	$(x+y)'$
0	0	1
0	1	0
1	0	0
1	1	0

Portes Logiques

Implémentation matérielle

- Les portes logiques simples peuvent être réalisées simplement à l'aide de quelques composants électroniques. Les portes logiques les plus communes sont également disponibles sous la forme de **circuits intégrés**.

- Exemple : circuit intégré **74HCT08** (quatre portes ET)



- Il existe également des circuits intégrés dans lesquels il est possible de programmer (avec un *Hardware Description Language* – HDL) des fonctions logiques plus ou moins complexes.

- Exemples :



Gate Array Logic (GAL)



Complex Programmable Logic Device (CPLD)

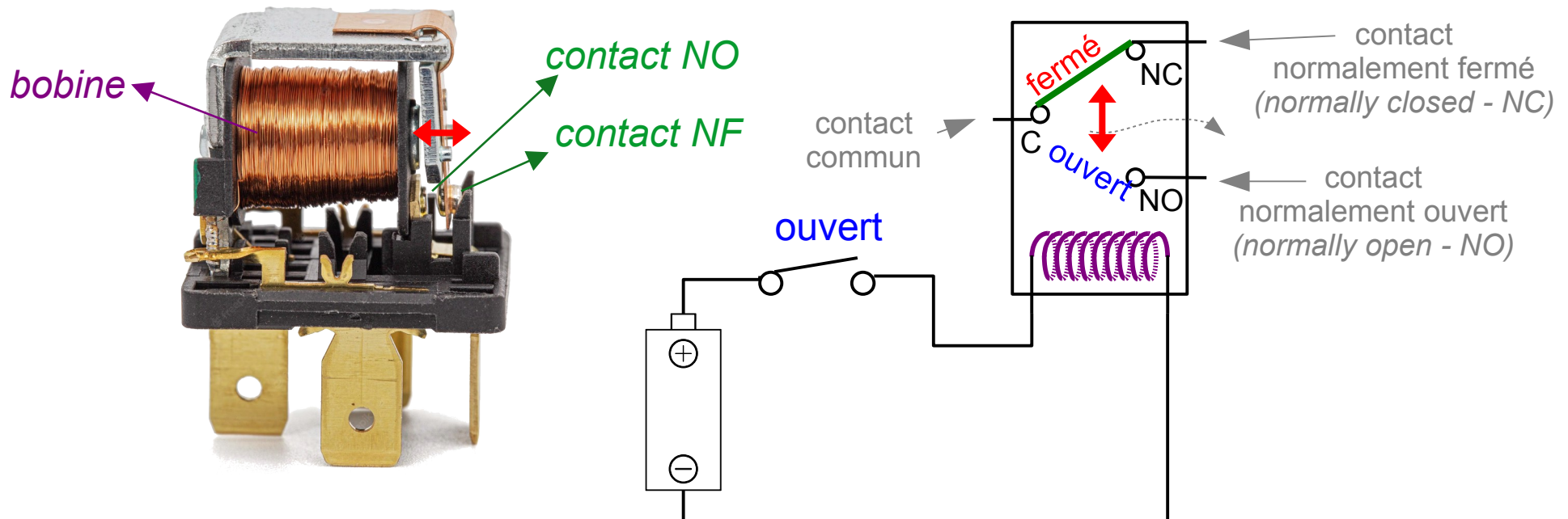


Field Programmable Gate Array (FPGA)

Portes Logiques

Intuition basée sur des circuits à relais

- Comment une porte logique est-elle implémentée ? Pour en donner l'intuition, nous utilisons une implémentation basée sur des relais.
- Un **relais** est un interrupteur commandé électriquement. Un champ magnétique est induit par une **bobine** lorsqu'elle est traversée par un courant. Ce champ magnétique déplace le **contact** de l'interrupteur.

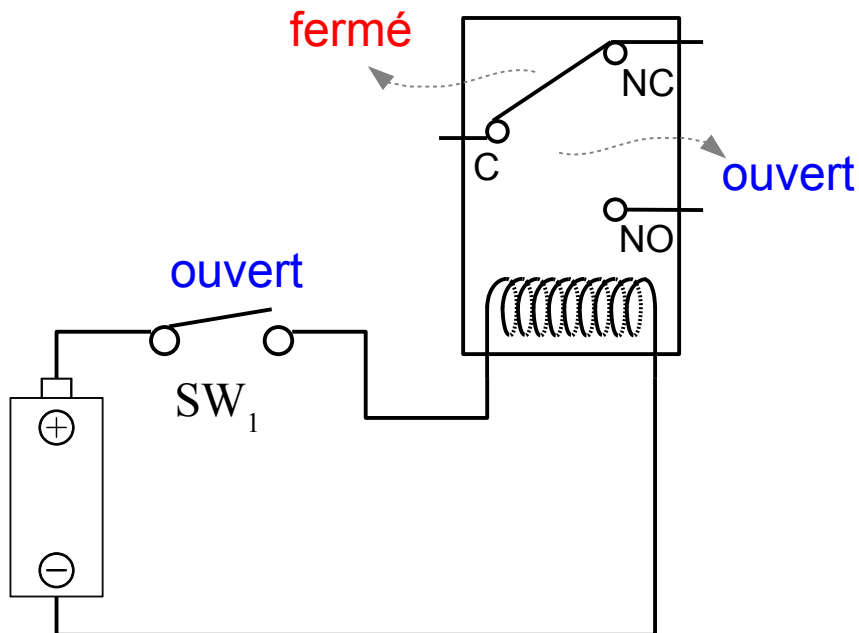


Portes Logiques

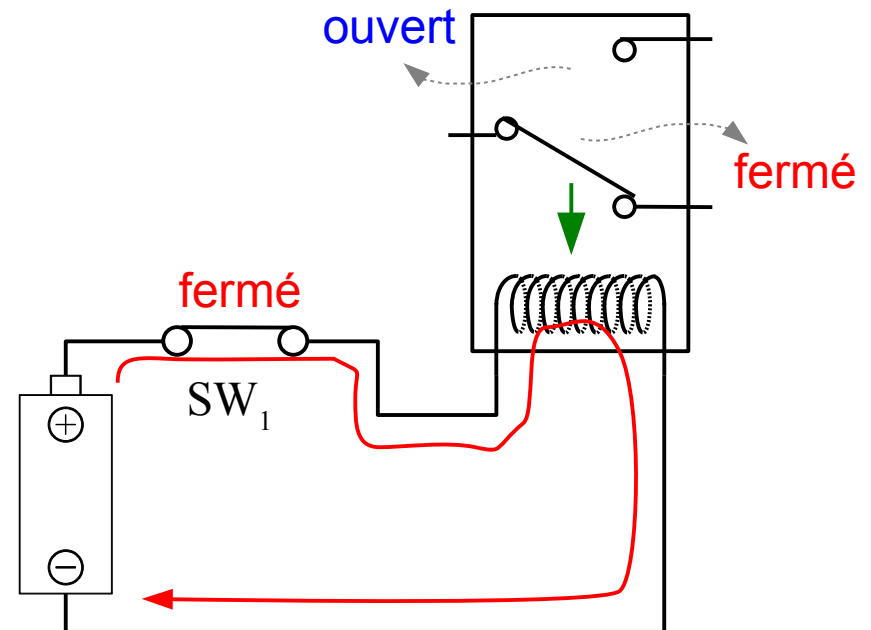
Intuition basée sur des circuits à relais

- Les exemples ci-dessous illustrent comment un circuit commande la fermeture ou l'ouverture de l'interrupteur du relais.

L'interrupteur SW_1 est ouvert, donc aucun courant ne circule dans la bobine du relais
→ l'interrupteur du relais est ouvert.



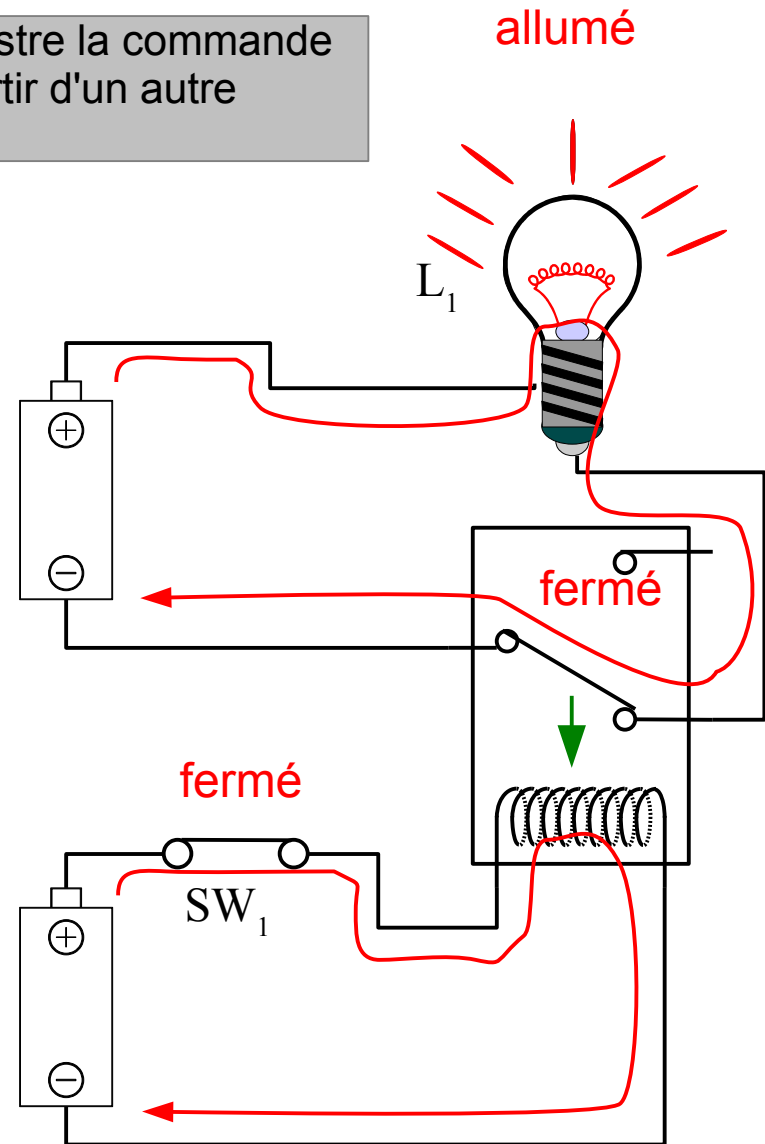
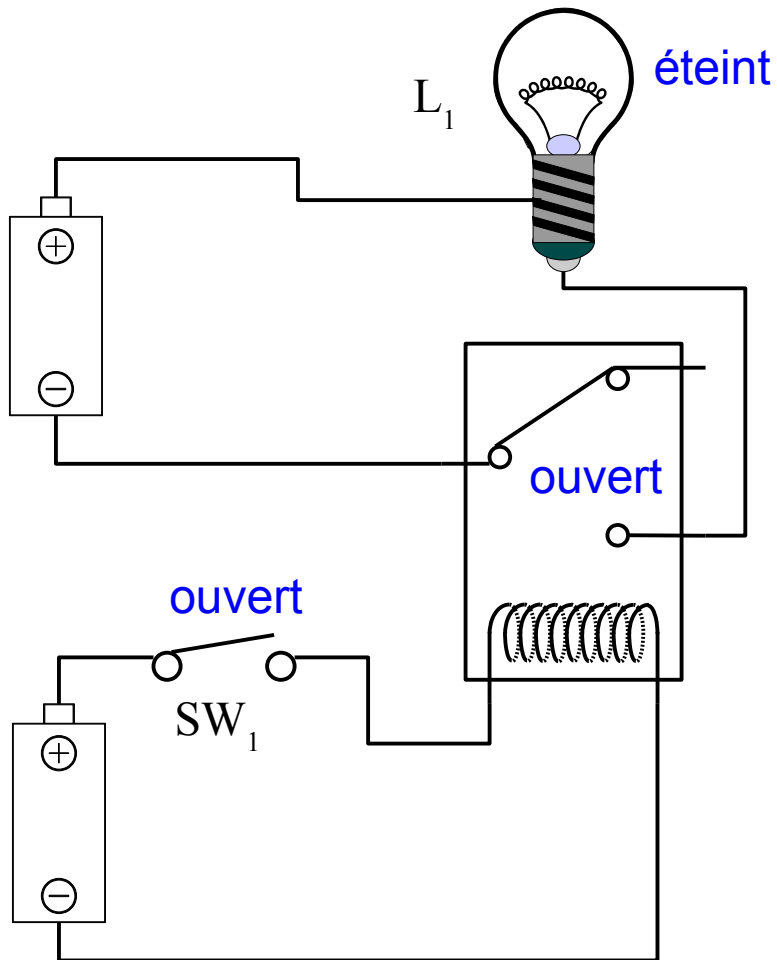
L'interrupteur SW_1 est fermé, un courant circule dans la bobine du relais
→ l'interrupteur du relais est fermé.



Portes Logiques

Circuits à relais

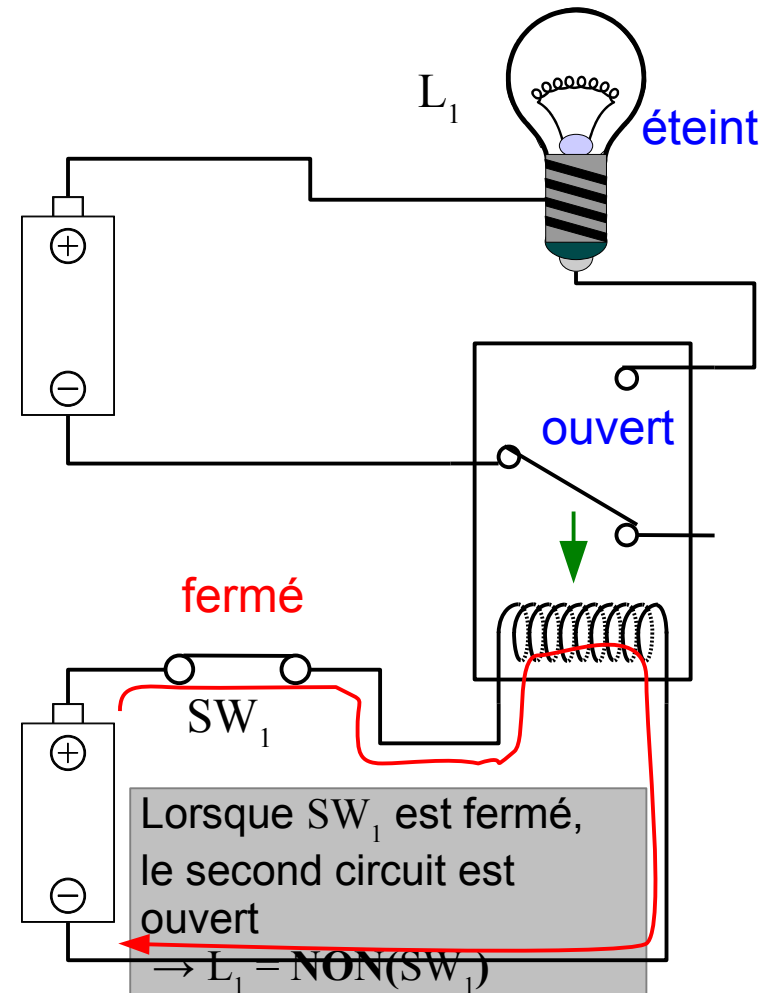
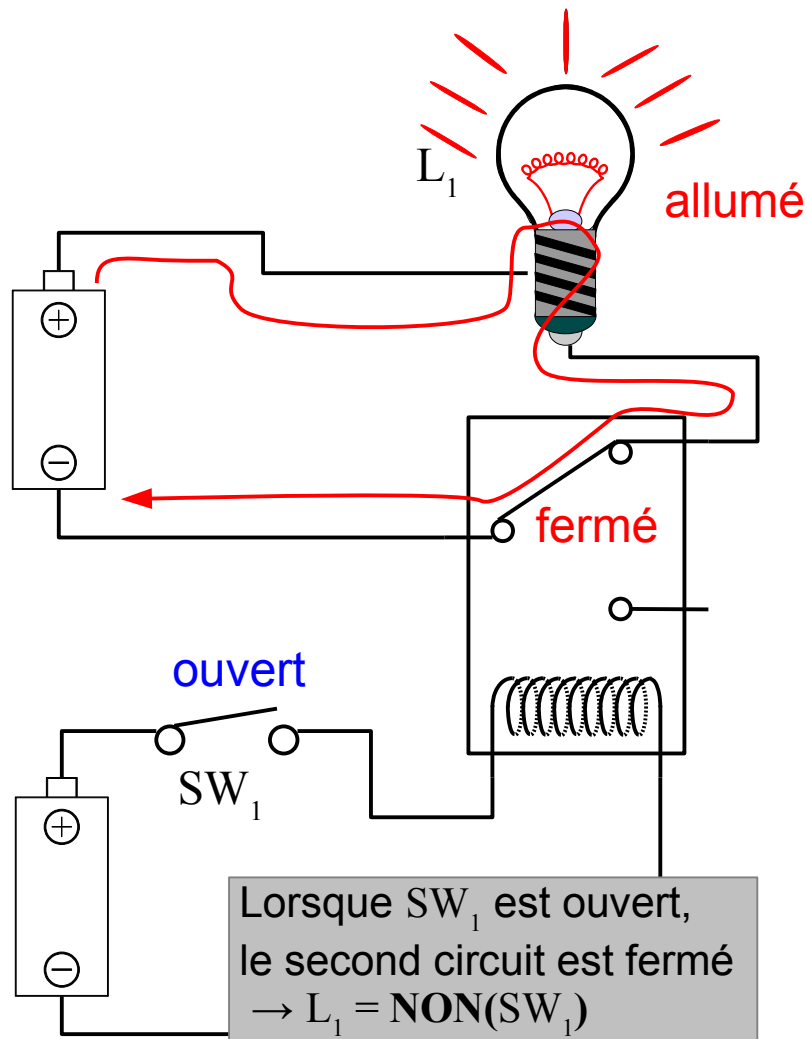
Cet exemple illustre la commande d'un circuit à partir d'un autre circuit.



Portes Logiques

Porte NON à relais

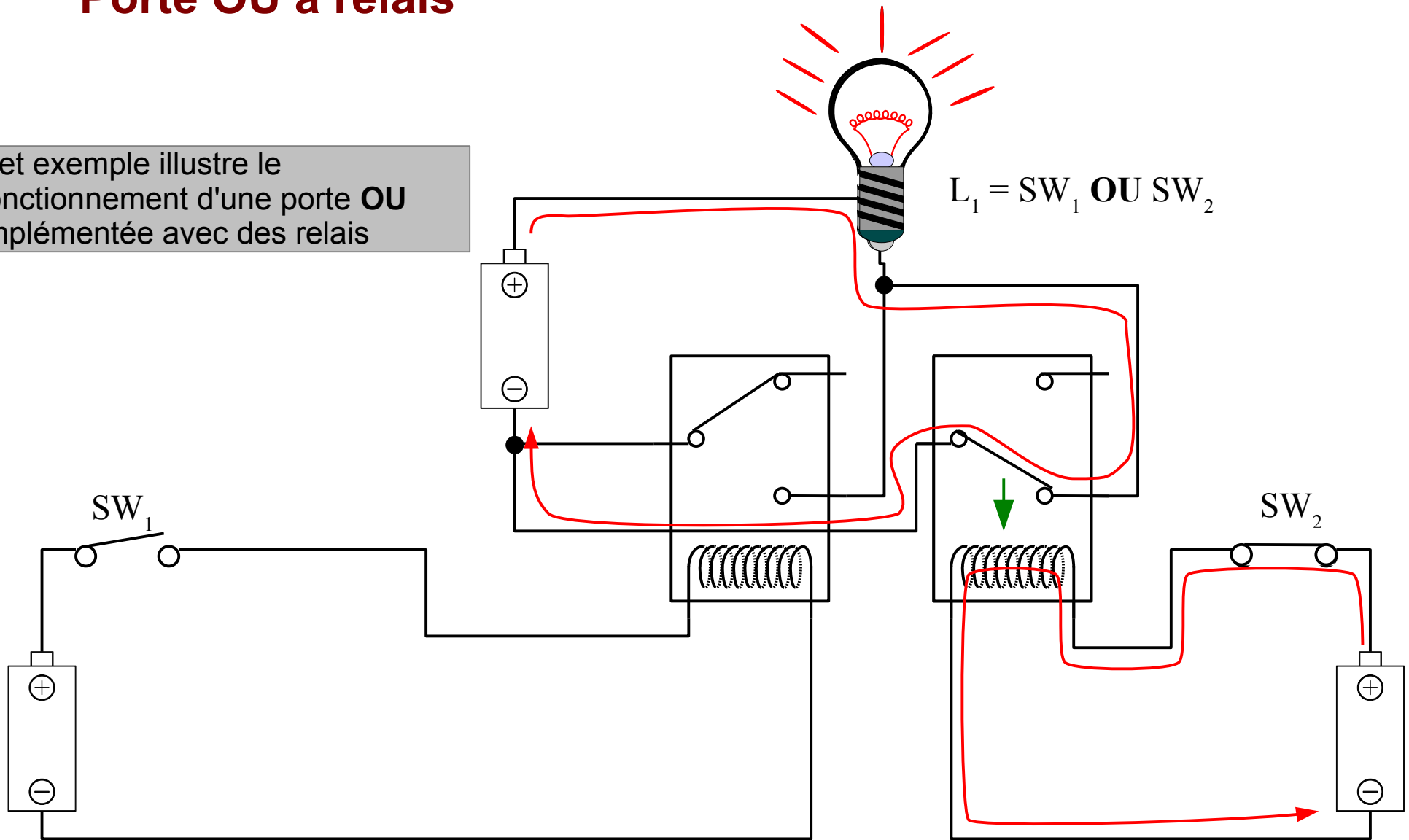
Cet exemple illustre le fonctionnement d'une porte **NON** implémentée avec un relais



Portes Logiques

Porte OU à relais

Cet exemple illustre le fonctionnement d'une porte **OU** implémentée avec des relais



Portes Logiques

Inconvénients des relais

- Les relais ont été utilisés dans les premiers ordinateurs. Ils ont un certain nombre d'inconvénients.
- Avec l'évolution de l'électronique, les relais ont donc vite été remplacés par des semi-conducteurs: *diodes*, *transistors bipolaires* (BJT), puis *transistors à effet de champ* (FET).

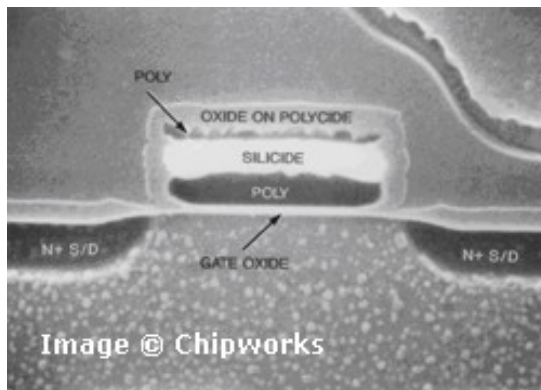
	Relais	Transistors
Temps de commutation (~vitesse)	millisecondes (10^{-3} s) (mouvements mécaniques)	picosecondes (10^{-12} s) - nanosecondes (10^{-9} s)
Problèmes mécaniques	Usure des contacts, rebonds	pas d'usure
Encombrement	$\sim \text{cm}^3$	nanomètres (10^{-9} m) quelques atomes
Consommation énergétique	élevée	très faible
Coût de production	élevé	très faible

Note : la taille d'un atome est de l'ordre d'un Ångström, i.e. 10^{-10} m.

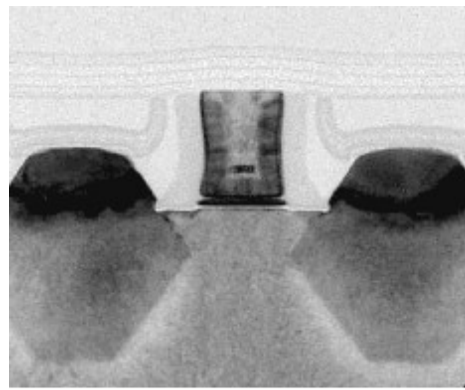
Portes Logiques

Utilisation de transistors

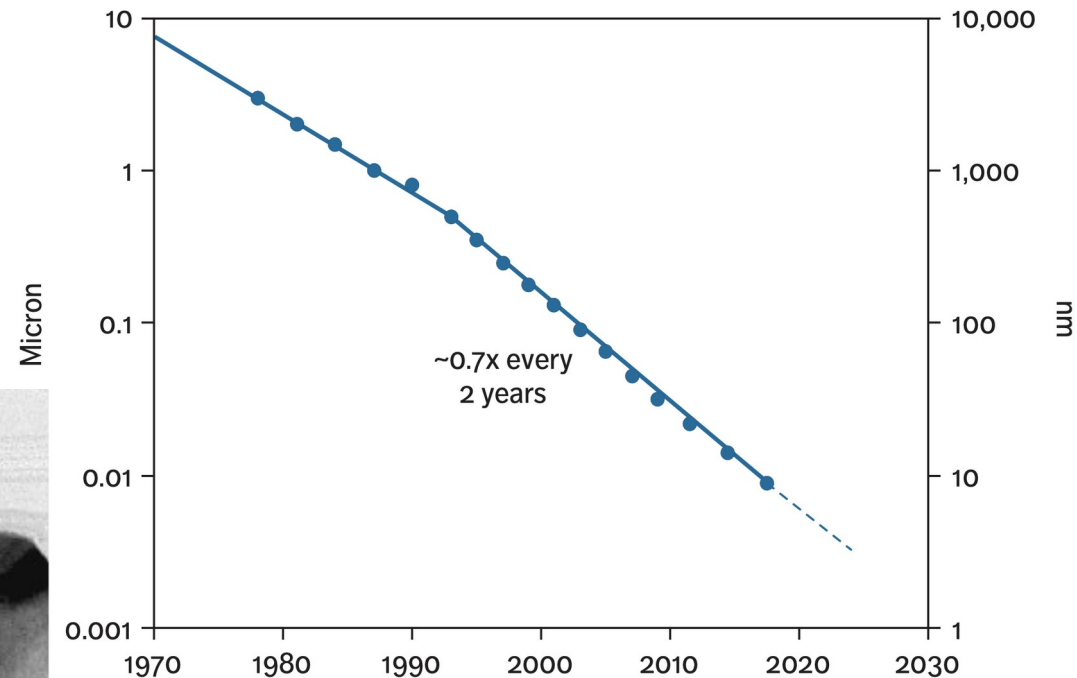
- Le **transistor** est un mécanisme électronique inventé en 1947 qui peut faire office d'interrupteur commandé électriquement.
- En électronique logique, les transistors sont de type "*à effet de champ*" et de technologie MOSFET (*Metal-Oxide Semiconductor Field Effect Transistor*).
- La taille des transistors (*feature size*) continue à diminuer...



Intel, 1992 (NMOS)
largeur de G: 1,1μm



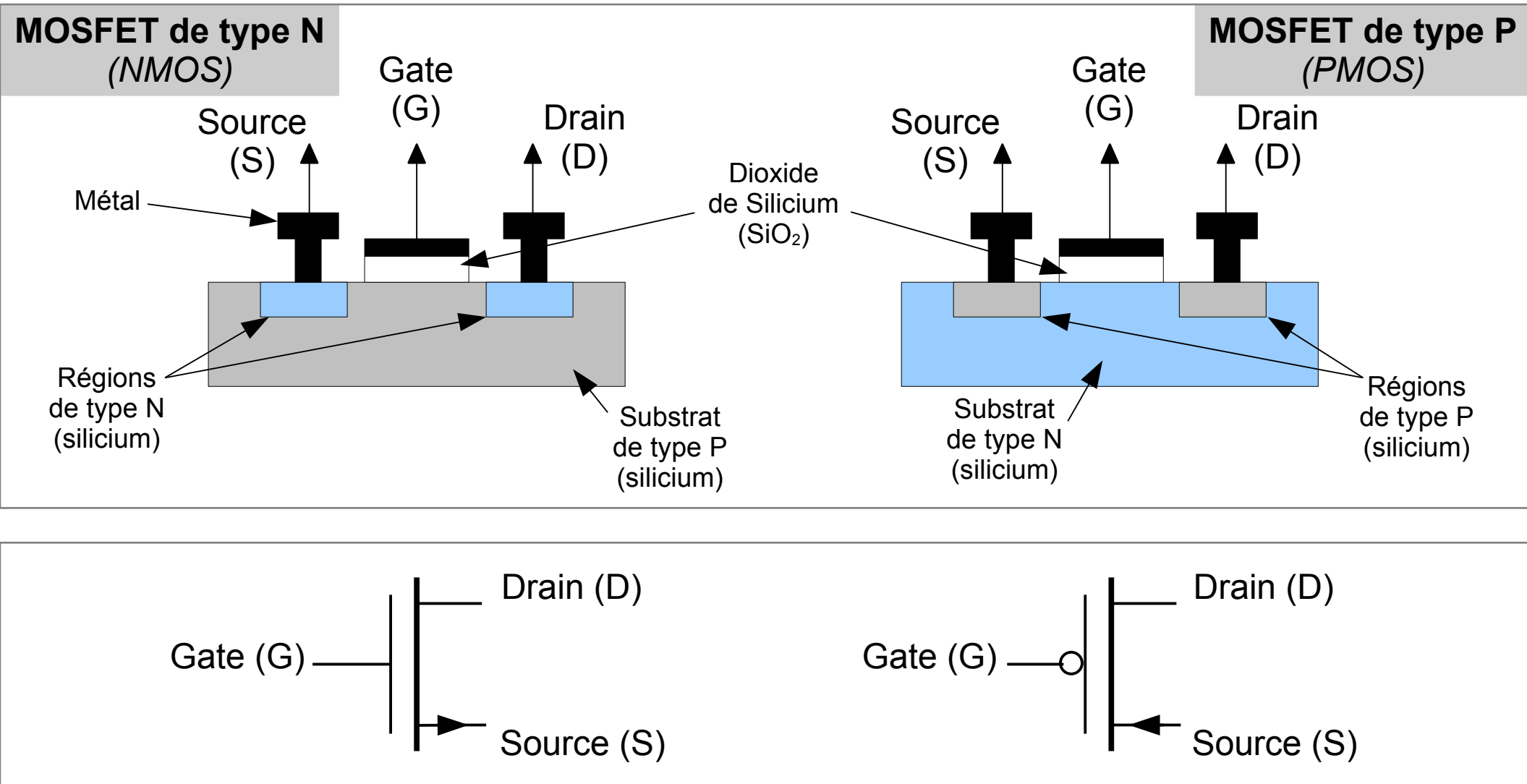
Intel, 2007 (PMOS)
largeur de G: 45 nm



M. T. Bohr and I. A. Young, "**CMOS Scaling Trends and Beyond**", IEEE Micro, 37(6), 2017

Implémentation

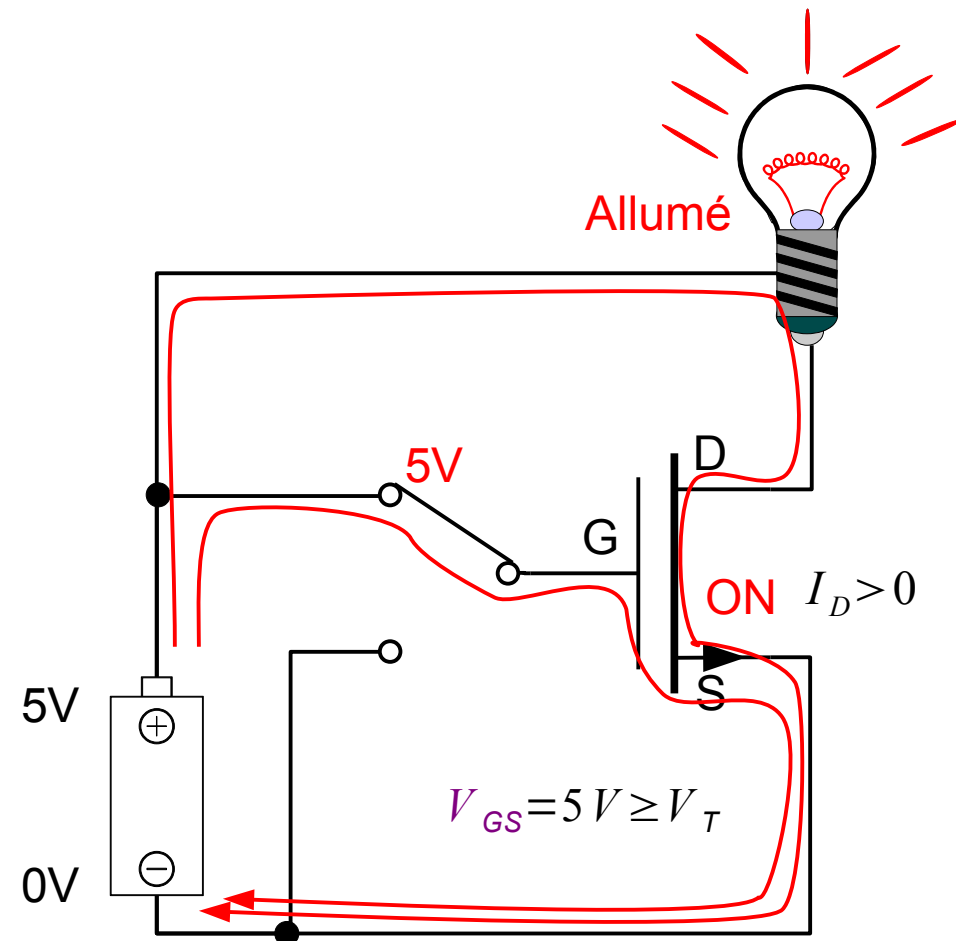
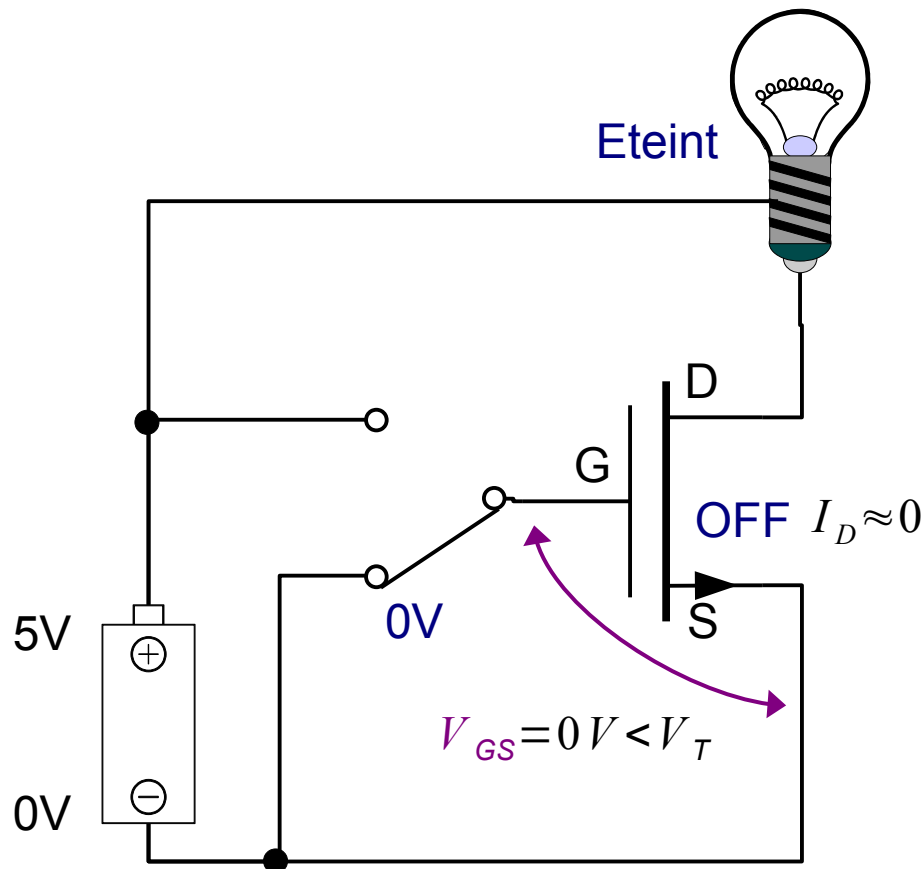
Transistors à effet de champ



Implémentation

Transistors à effet de champ

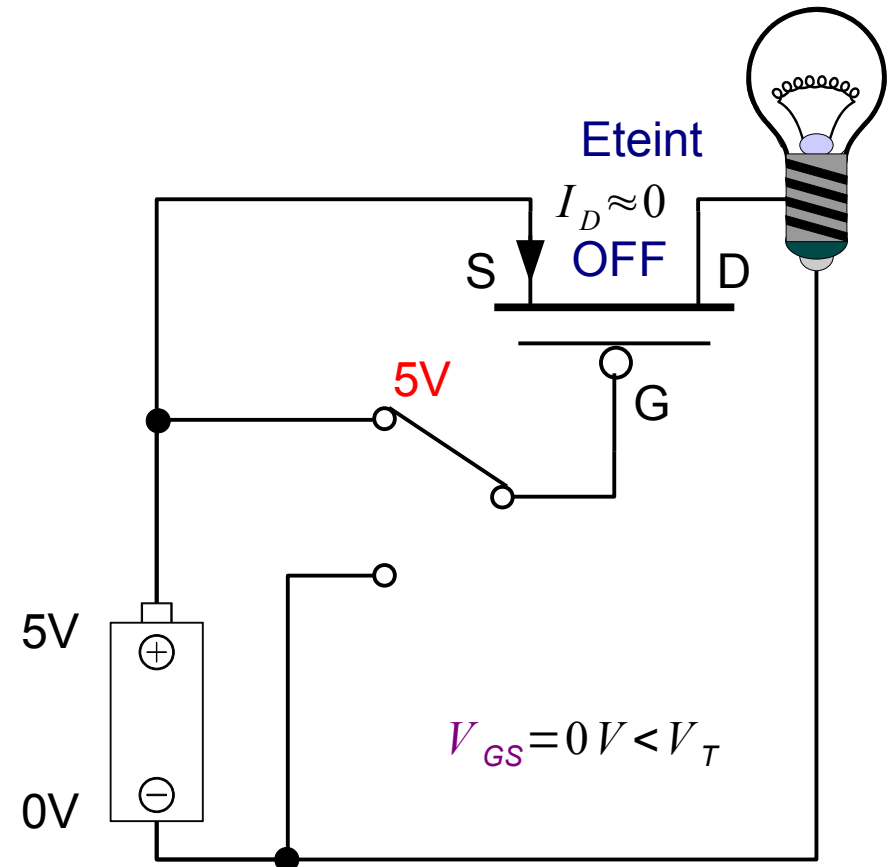
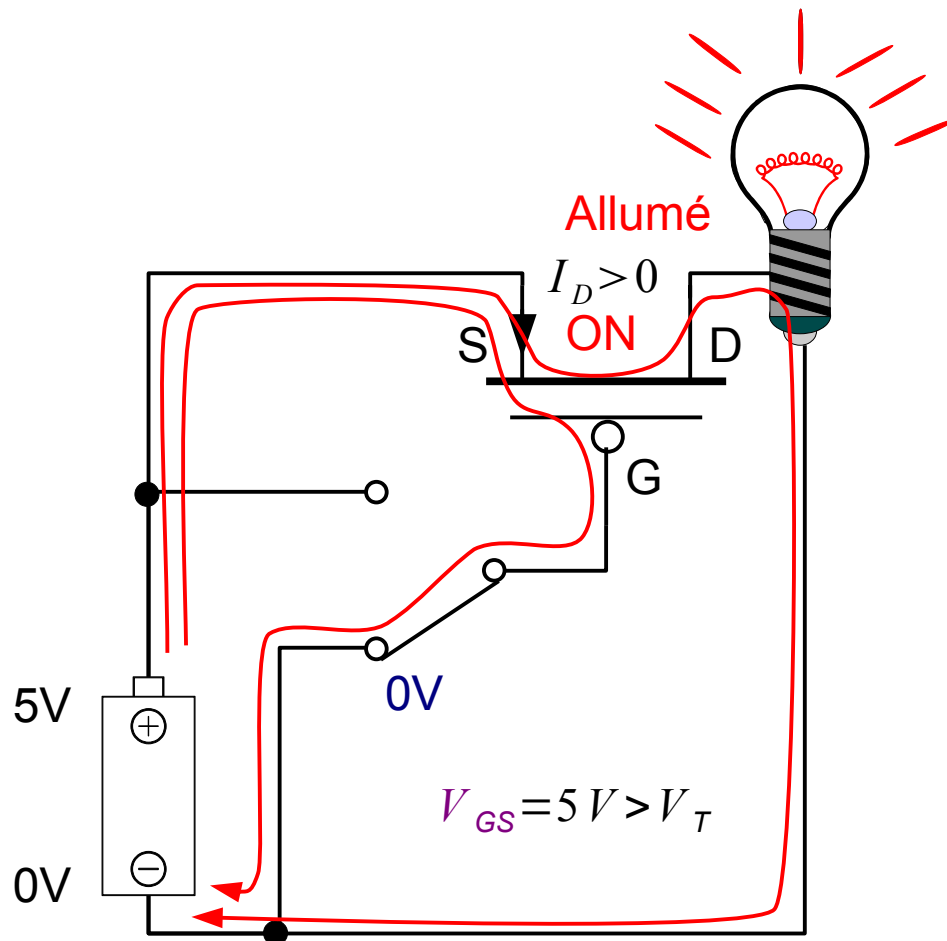
- Commutation avec MOSFET de **type N** (*enhancement mode*)
- Transistor passant si $V_{GS} > \text{seuil } V_T$



Implémentation

Transistors à effet de champ

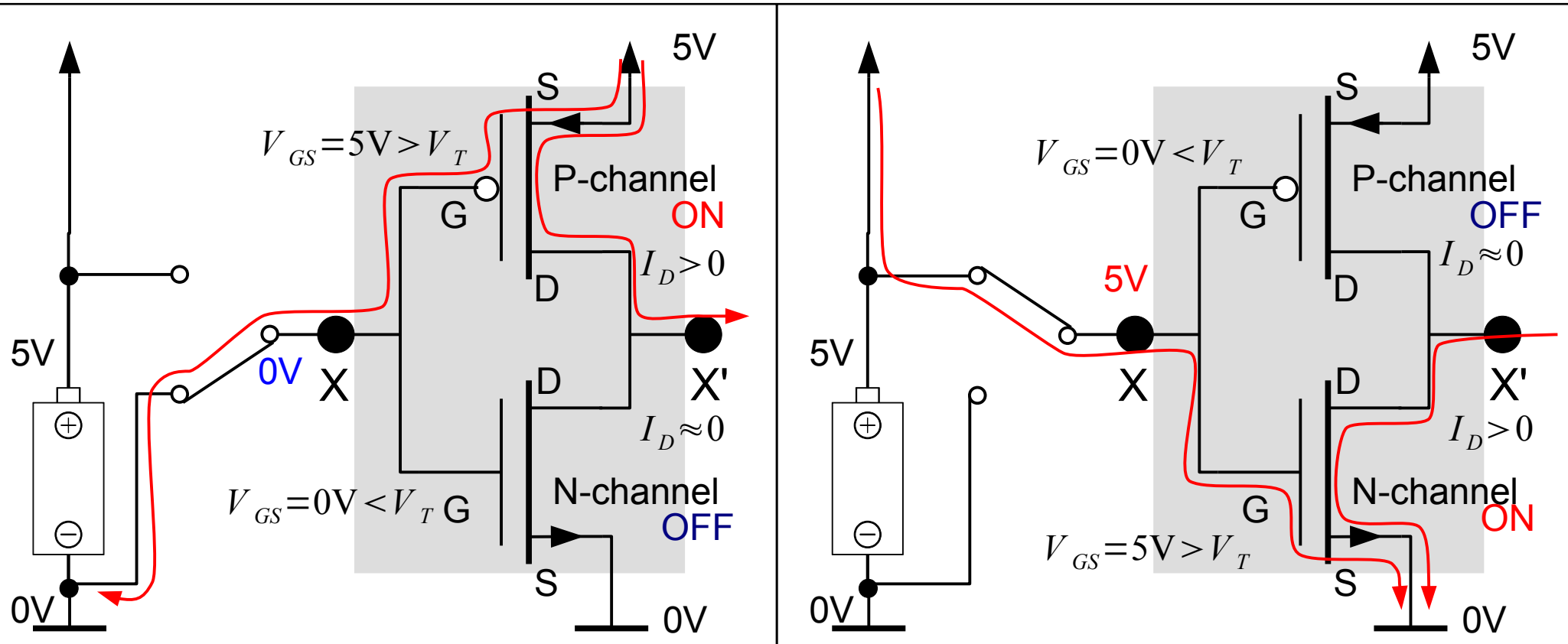
- Commutation avec MOSFET de **type P** (*enhancement mode*)
- Transistor passant si $V_{GS} > \text{seuil } V_T$



Implémentation

Technologie CMOS

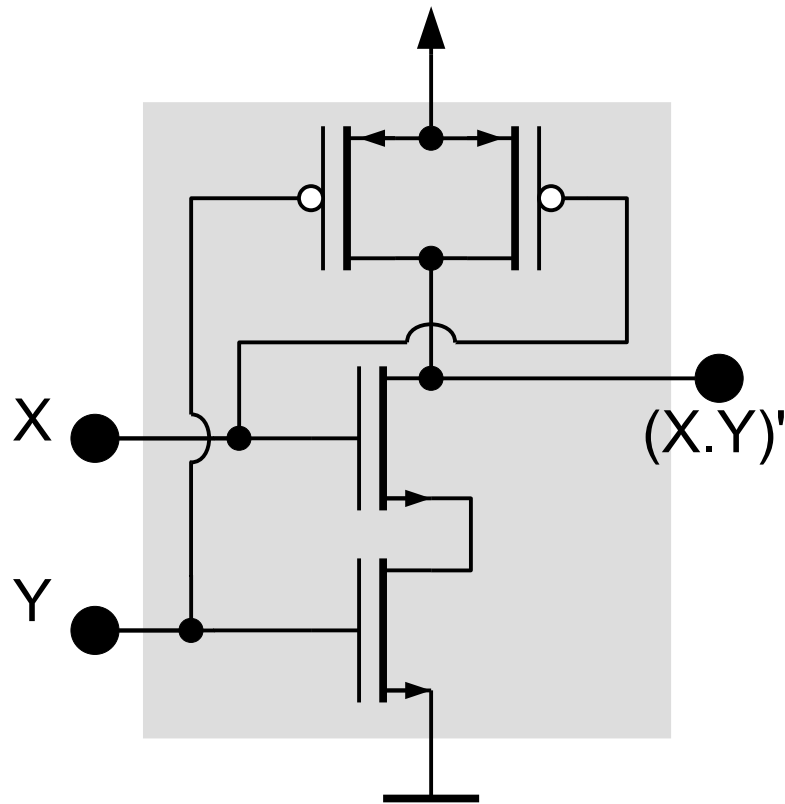
- La technologie de circuits logiques **CMOS** (*Complementary Metal-Oxide Semiconductor*) utilise des transistors MOSFETs de type **N** et **P**.
- Exemple : porte NON



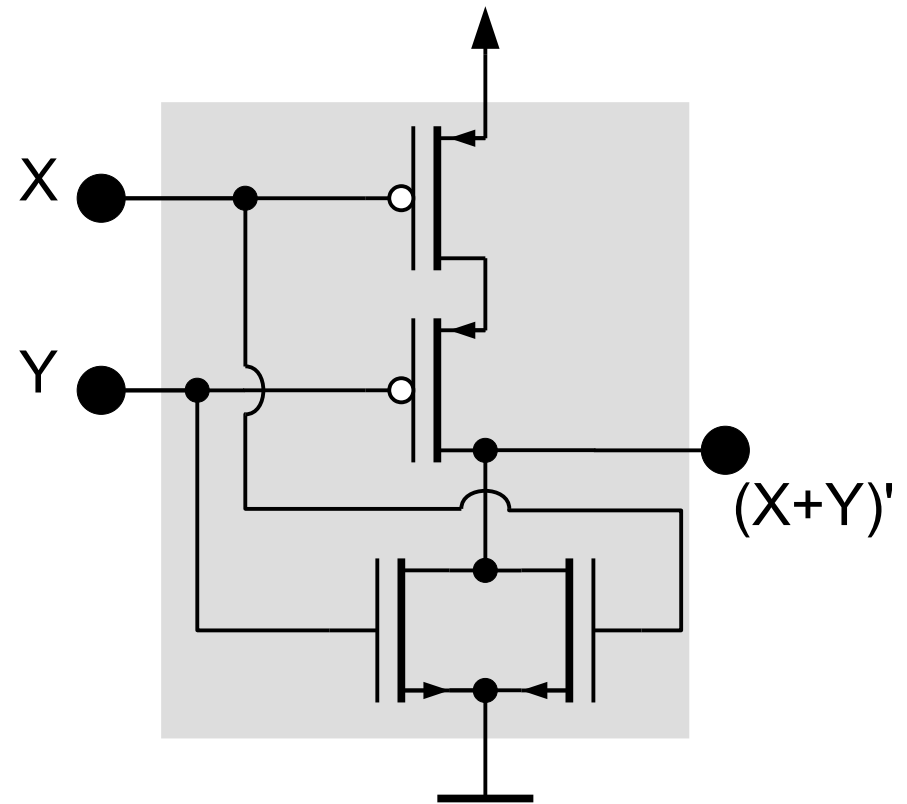
Implémentation

Portes Logiques CMOS

- En technologie CMOS, les portes **NAND** et **NOR** sont plus simples à implémenter que AND et OR. Les portes **NAND** et **NOR** sont **universelles** : elles permettent d'implémenter toutes les autres portes logiques à elles seules.



Porte NON-ET (NAND)



Porte NON-OU (NOR)

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

Principes

- Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Logique Combinatoire

Principes

- La **logique combinatoire** (*combinational logic*) consiste à combiner des portes logiques entre elles afin d'exprimer d'autres fonctions logiques (en général plus complexes).
- Un **circuit logique** est la combinaison de plusieurs portes.
- Propriété : En logique combinatoire, la sortie d'une combinaison de portes (résultat) dépend uniquement de l'état actuel des entrées (variables).

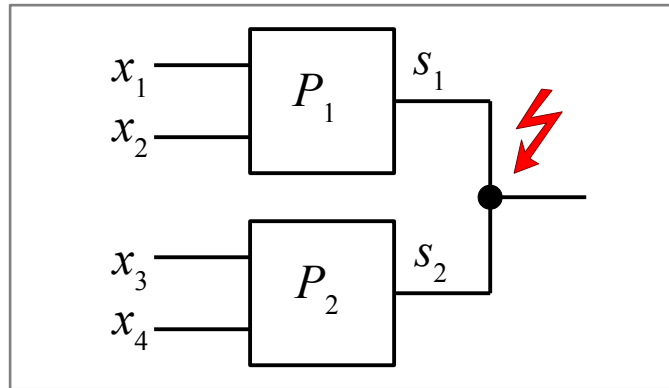
Objectifs

- Comment déterminer un circuit logique à partir d'une expression logique ? et vice-versa ? (table de vérité, somme canonique, tableaux de Karnaugh)
- Quels sont les blocs utiles de la logique combinatoire ?

Logique Combinatoire

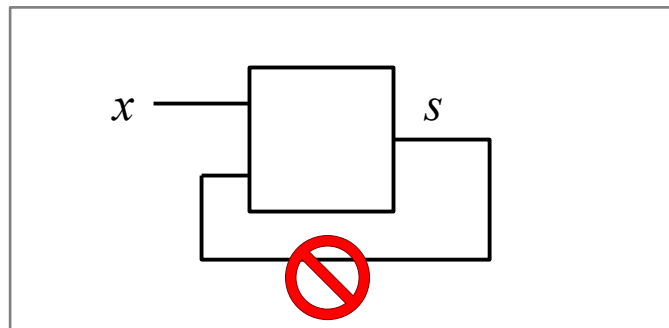
Règles des circuits logiques

- Deux sorties ne peuvent pas être directement connectées entre elles.



Supposons que
• $s_1 = 0$ (0V)
• $s_2 = 1$ (5V)
→ **court-circuit**

- Pas de boucle : il n'est pas permis de connecter une sortie d'une porte en aval à l'entrée d'une porte en amont.

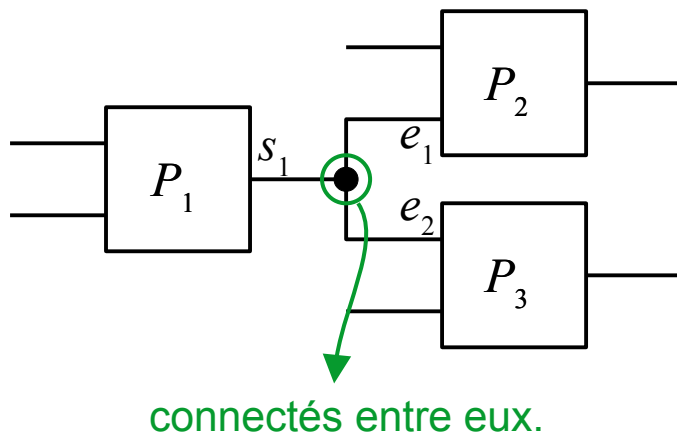


Les boucles sont utilisées en logique séquentielle pour garder un état ou créer un oscillateur (état change périodiquement).

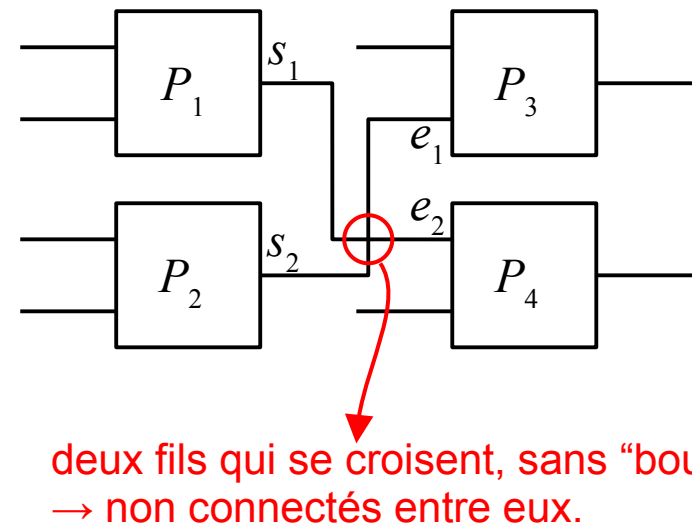
Logique Combinatoire

Règles des circuits logiques

- Convention pour les connexions
 - utilisation de “boules noires” pour montrer quand deux fils qui se croisent sont connectés entre eux



la sortie s_1 est connectée
aux entrées e_1 et e_2

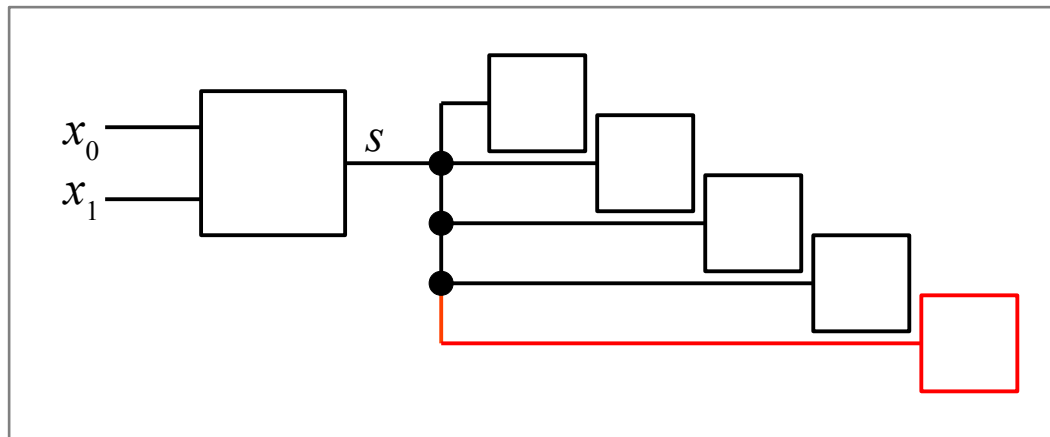


la sortie s_1 est connectée à l'entrée e_2
la sortie s_2 est connectée à l'entrée e_1

Logique Combinatoire

Contraintes physiques

- Une sortie de porte logique ne peut pas être connectée à un nombre arbitrairement grand d'entrées. Le nombre de portes logiques en aval à laquelle une sortie peut être connectée est limité. Ce nombre est appelé le **fan-out**.



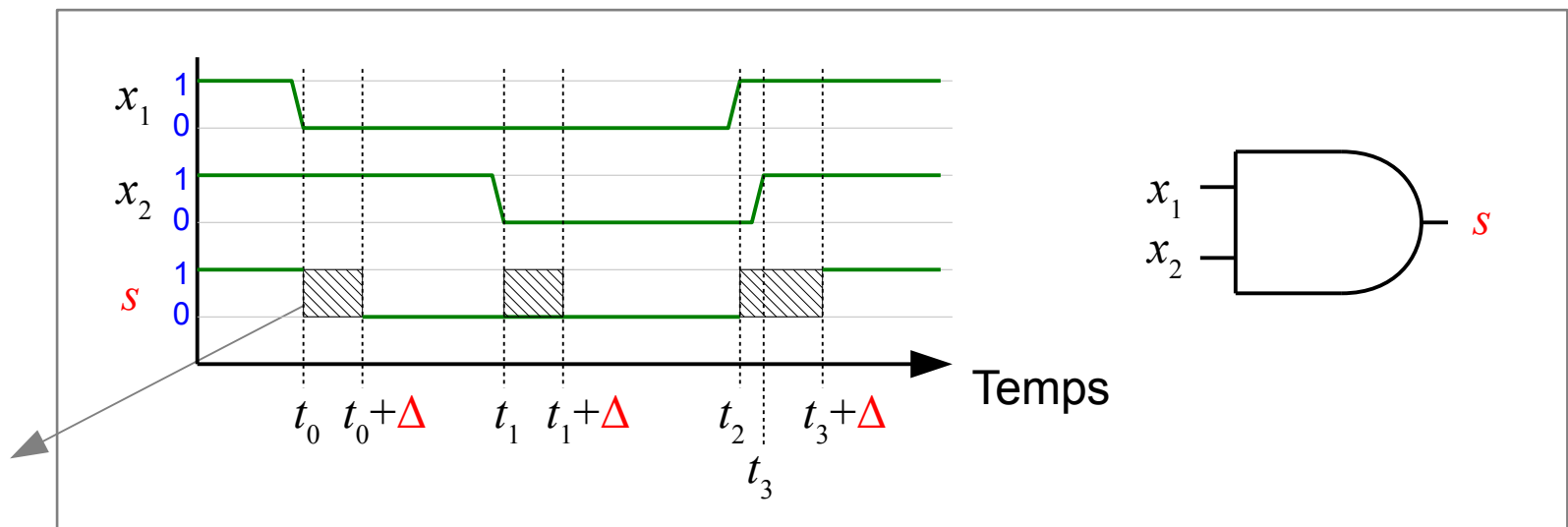
Chaque porte connectée à une sortie constitue une “charge” supplémentaire pour cette sortie.

- 1). Au delà d'une certaine charge (d'un certain courant à fournir en sortie), la tension électrique en sortie risque de ne plus correspondre à l'état 0 ou 1.
- 2). la charge peut également avoir un impact sur le délai de propagation de la porte.

Logique Combinatoire

Contraintes physiques

- Contrairement aux fonctions logiques qu'elles implémentent, les portes logiques ne réagissent pas instantanément à un changement de leurs entrées. Le résultat en sortie ne sera correct qu'après un **délai de propagation Δ** .



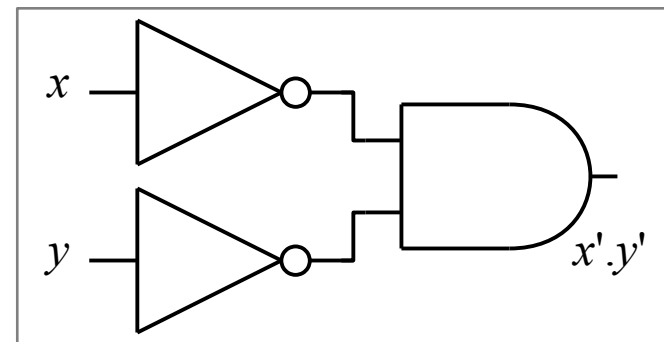
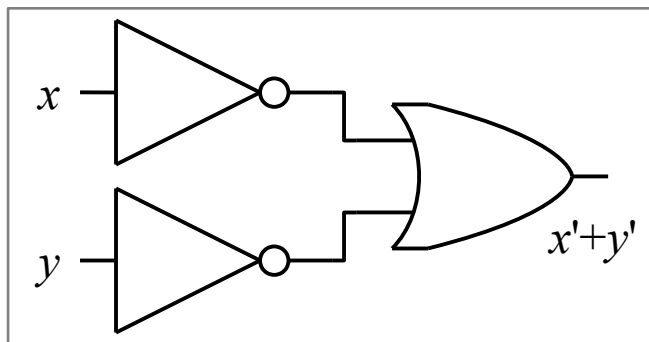
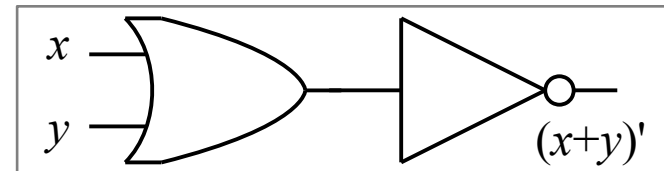
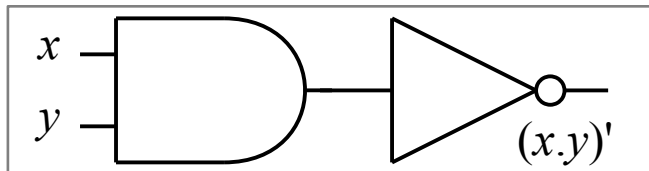
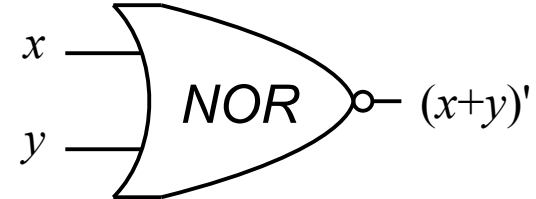
Cette zone correspond à un moment où il n'est pas garanti que la sortie soit dans le bon état.

- Le délai de propagation dépend notamment de la façon dont la porte logique est implémentée. Ce délai est typiquement de l'ordre de quelques nanosecondes. Il diffère d'une porte à l'autre.

Logique Combinatoire

Combinaison de portes

- Il est possible de construire une porte logique avec d'autres portes logiques.
- Exemple : construction des portes **NON-ET (NAND)** et **NON-OU (NOR)**.

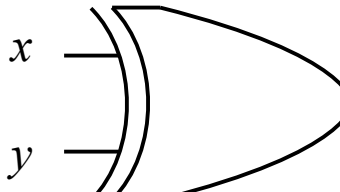


Les 2 implémentations
sont équivalentes par
De Morgan (T13)

Logique Combinatoire

Autres fonctions logiques

- Question : Comment exprimer d'autres fonctions logiques que ET, OU, NON, NON-ET et NON-OU ?
- Exemple : comment exprimer la fonction **OU-exclusif (XOR)** ?
Cette fonction ne donne un résultat vrai que si une seule entrée est vraie à la fois.

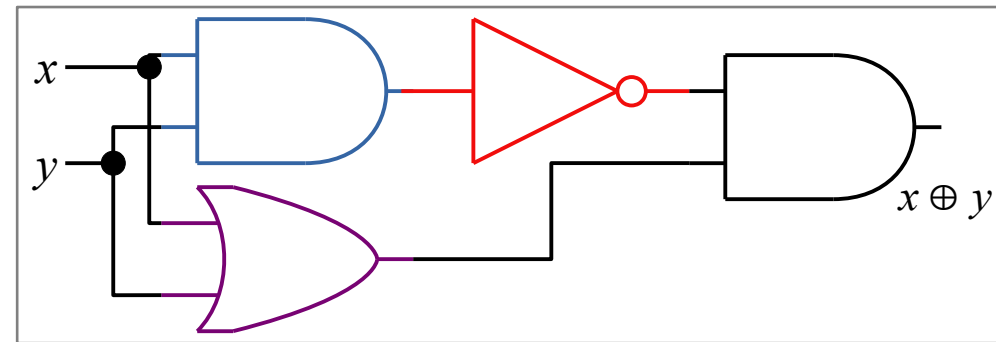
OU-EXCLUSIF (XOR)																	
	$F(x, y) = x \oplus y$	<table><tr><th>x</th><th>y</th><th>$F(x, y)$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	$F(x, y)$	0	0	0	0	1	1	1	0	1	1	1	0
x	y	$F(x, y)$															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

Logique Combinatoire

Autres fonctions logiques

- Première approche, exprimer la fonction : XOR est vrai si x est vrai ou si y est vrai, mais **pas** lorsque x et y sont vrais simultanément.

$$x \oplus y = (x + y) \cdot (x \cdot y)'$$



- Seconde approche, dériver une expression logique à partir de la table de vérité.

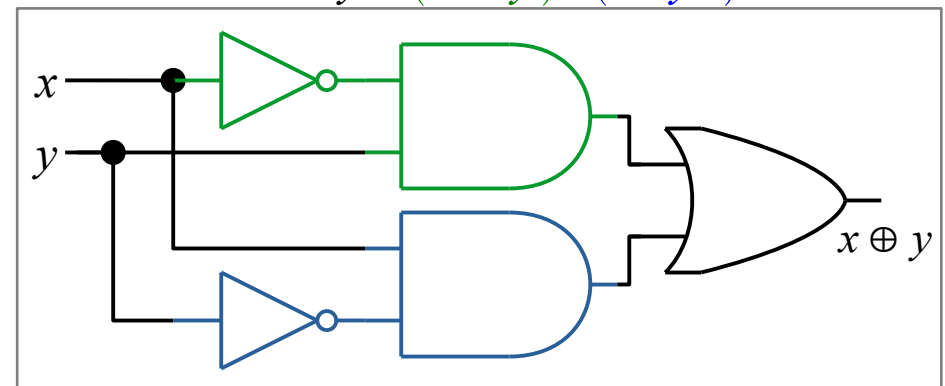
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

→ $(x' \cdot y)$

→ $(x \cdot y')$

“somme canonique”

$$x \oplus y = (x' \cdot y) + (x \cdot y')$$



Logique Combinatoire

Somme (ou produit) canonique

- L'exemple précédent illustre un moyen générique de développer une fonction booléenne sous forme d'une somme (ou produit) de termes.
- Point de départ
 - Toute fonction $f(x)$ d'une variable peut être écrite comme $f(x) = ax + bx'$
 - si $x = 1$, alors $f(1) = a \cdot 1 + b \cdot 0 = a$
 - si $x = 0$, alors $f(0) = a \cdot 0 + b \cdot 1 = b$
 - d'où $f(x) = x \cdot f(1) + x' \cdot f(0)$
- Fonction de $N=2$ variables (et plus...)
 - Par un raisonnement similaire, on peut exprimer $f(x,y)$ comme suit
$$f(x,y) = x \cdot f(1,y) + x' \cdot f(0,y)$$
$$= \underbrace{x \cdot y}_{\text{minterms}} \cdot f(1,1) + \underbrace{x \cdot y'}_{\text{minterms}} \cdot f(1,0) + \underbrace{x' \cdot y}_{\text{minterms}} \cdot f(0,1) + \underbrace{x' \cdot y'}_{\text{minterms}} \cdot f(0,0)$$

minterms : termes produit normalisés des N variables ou de leur complément

Logique Combinatoire

Somme (ou produit) canonique

- Cette technique permet d'exprimer n'importe quelle table de vérité sous la forme d'une **somme canonique** (*forme normale disjonctive*) ou d'un **produit canonique** (*forme normale conjonctive*).
- Principe :
 - Un produit (resp. somme) normalisé de N variables, appelé *minterm* (resp. *maxterm*) ne prend la valeur **1** (**0**) que pour une unique assignation de valeurs aux N variables.
 - Exemple :

x	y	z	u	<i>minterm</i>	<i>maxterm</i>
1	1	0	1	$x.y.z'.u = 1$	$x'+y'+z+u' = 0$

- A chaque ligne de la table de vérité, on peut donc faire correspondre un unique *minterm* et un unique *maxterm*.
- Une somme (resp. produit) canonique est un produit de *minterms* (resp. *maxterms*)

Logique Combinatoire

Somme et produit canoniques

- Chaque ligne k de la table de vérité de la fonction $F(x,y,z)$ ci-dessous est complétée avec la paire de *minterm* et *maxterm* qui lui est associée.

	x	y	z	$F(x,y,z)$	<i>minterms</i>	<i>maxterms</i>
0	0	0	0		$x'.y'.z'$	$x+y+z$
1	0	0	1		$x'.y'.z$	$x+y+z'$
2	0	1	0		$x'.y.z'$	$x+y'+z$
3	0	1	1		$x'.y.z$	$x+y'+z'$
4	1	0	0		$x.y'.z'$	$x'+y+z$
5	1	0	1		$x.y'.z$	$x'+y+z'$
6	1	1	0		$x.y.z'$	$x'+y'+z$
7	1	1	1		$x.y.z$	$x'+y'+z'$

minterms et maxterms
don't indépendants
de la fonction !

Le *minterm* $x'.y.z'$ vaut
1 à la ligne 2 et
0 dans toutes les autres.

Le *maxterm* $x+y'+z'$ vaut
0 à la ligne 3 et
1 dans toutes les autres.

Logique Combinatoire

Somme et produit canoniques

- La correspondance entre la table de vérité d'une fonction logique $F(x_1, \dots, x_N)$ et des *minterms* / *maxterms* permet d'en dériver une expression logique.
- Somme canonique** = somme des *minterms* qui correspondent aux lignes de la table de vérité où $F(x_1, \dots, x_N) = 1$.

$$F(x_1, \dots, x_4) = \sum_{x_1, \dots, x_4} (0, 3, 7, 8, 11)$$

Liste des lignes de la table de vérité où la fonction vaut 1.

- Produit canonique** = produit des *maxterms* qui correspondent aux lignes de la table de vérité où $F(x_1, \dots, x_N) = 0$.

$$F(x_1, \dots, x_4) = \prod_{x_1, \dots, x_4} (1, 2, 4, 5, 6, 9, 10, 12, 13, 14, 15)$$


Liste des lignes de la table de vérité où la fonction vaut 0.

Logique Combinatoire

Somme canonique

- Exemple : $F(x,y,z)$ est une fonction choisie arbitrairement.

	x	y	z	$F(x,y,z)$	<i>minterms</i>	<i>maxterms</i>
0	0	0	0	1	$x'.y'.z'$	$x+y+z$
1	0	0	1	0	$x'.y'.z$	$x+y+z'$
2	0	1	0	0	$x'.y.z'$	$x+y'+z$
3	0	1	1	1	$x'.y.z$	$x+y'+z'$
4	1	0	0	1	$x.y'.z'$	$x'+y+z$
5	1	0	1	0	$x.y'.z$	$x'+y+z'$
6	1	1	0	1	$x.y.z'$	$x'+y'+z$
7	1	1	1	1	$x.y.z$	$x'+y'+z'$

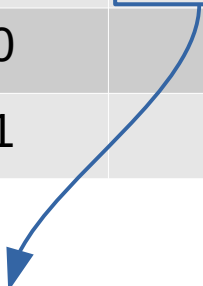

$$\begin{aligned} F(x, y, z) &= \sum_{x,y,z} (0,3,4,6,7) \\ &= x'.y'.z' + x'.y.z + x.y'.z' + x.y.z' + x.y.z \end{aligned}$$

Logique Combinatoire

Produit canonique

- Exemple : $F(x,y,z)$ est une fonction choisie arbitrairement.

	x	y	z	$F(x,y,z)$	<i>minterms</i>	<i>maxterms</i>
0	0	0	0	1	$x'.y'.z'$	$x+y+z$
1	0	0	1	0	$x'.y'.z$	$x+y+z'$
2	0	1	0	0	$x'.y.z'$	$x+y'+z$
3	0	1	1	1	$x'.y.z$	$x+y'+z'$
4	1	0	0	1	$x.y'.z'$	$x'+y+z$
5	1	0	1	0	$x.y'.z$	$x'+y+z'$
6	1	1	0	1	$x.y.z'$	$x'+y'+z$
7	1	1	1	1	$x.y.z$	$x'+y'+z'$


$$\begin{aligned} F(x, y, z) &= \prod_{x, y, z} (1, 2, 5) \\ &= (x+y+z') \cdot (x+y'+z) \cdot (x'+y+z') \end{aligned}$$

Logique Combinatoire

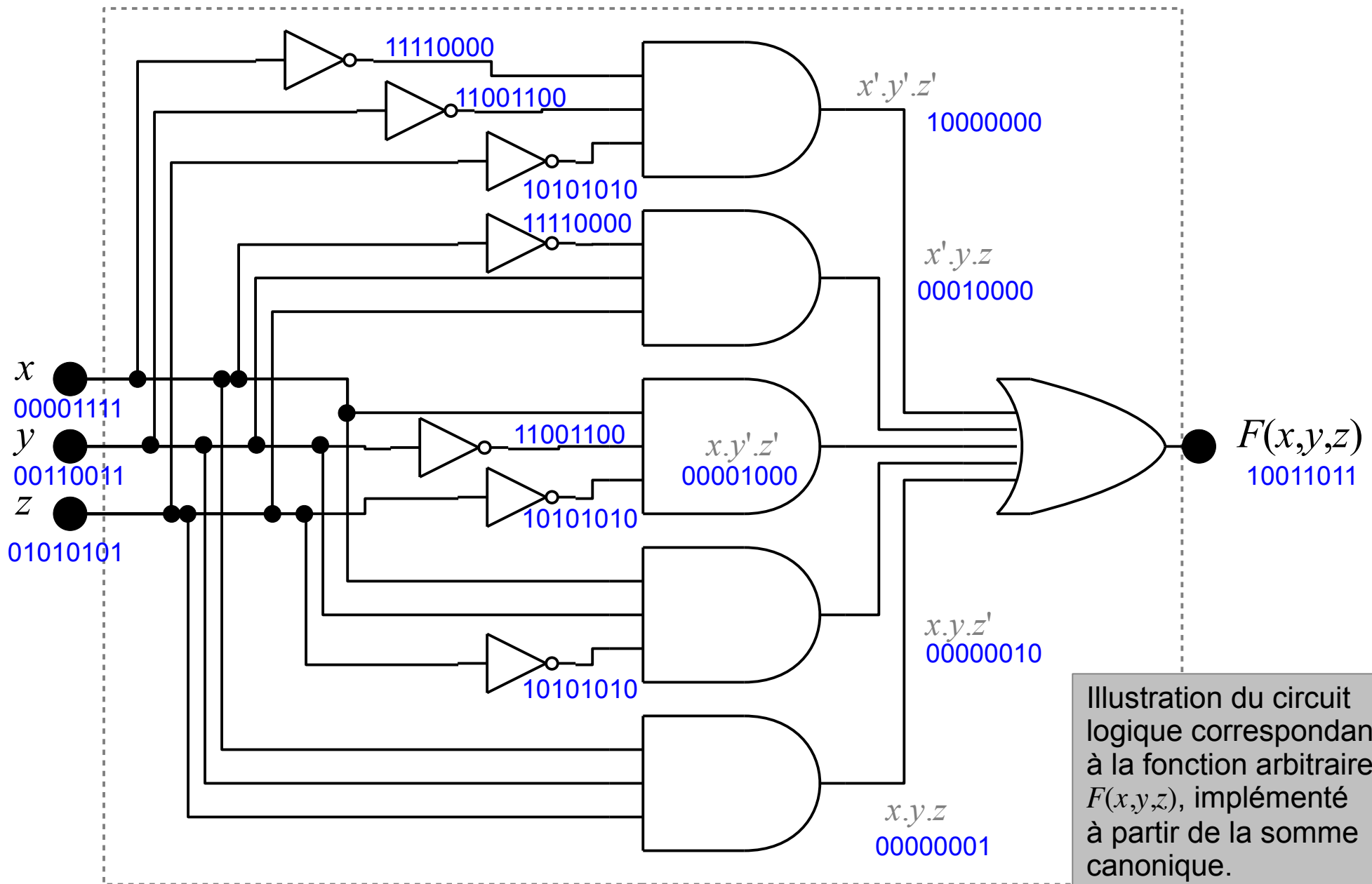
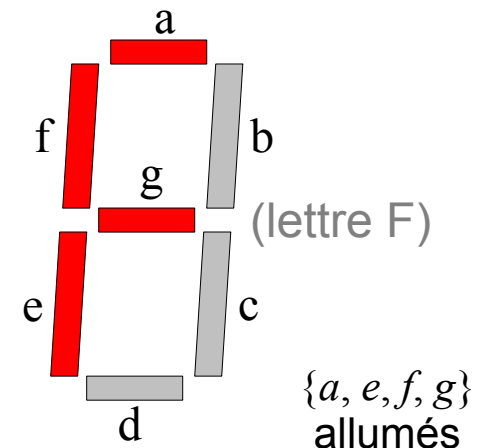
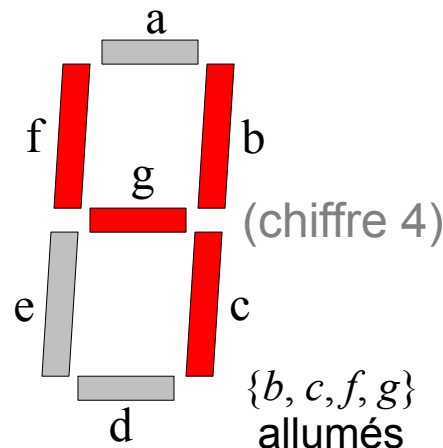
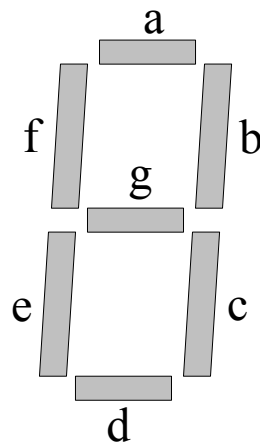


Illustration du circuit logique correspondant à la fonction arbitraire $F(x,y,z)$, implémenté à partir de la somme canonique.

Logique Combinatoire

Somme et produit canoniques

- Exercice : Afficheur à 7 segments (question d'examen d'août 2012)
 - Un afficheur à 7 segments est un dispositif électronique composé de 7 segments lumineux (notés $\{a, b, \dots, g\}$) qui peuvent être allumés / éteints individuellement.



- Etablir une fonction logique donnant la valeur du segment 'e' en fonction d'un code c de 4 bits qui représente le chiffre hexadécimal à afficher.
- Utiliser à cet effet une somme ou un produit canonique.

Logique Combinatoire

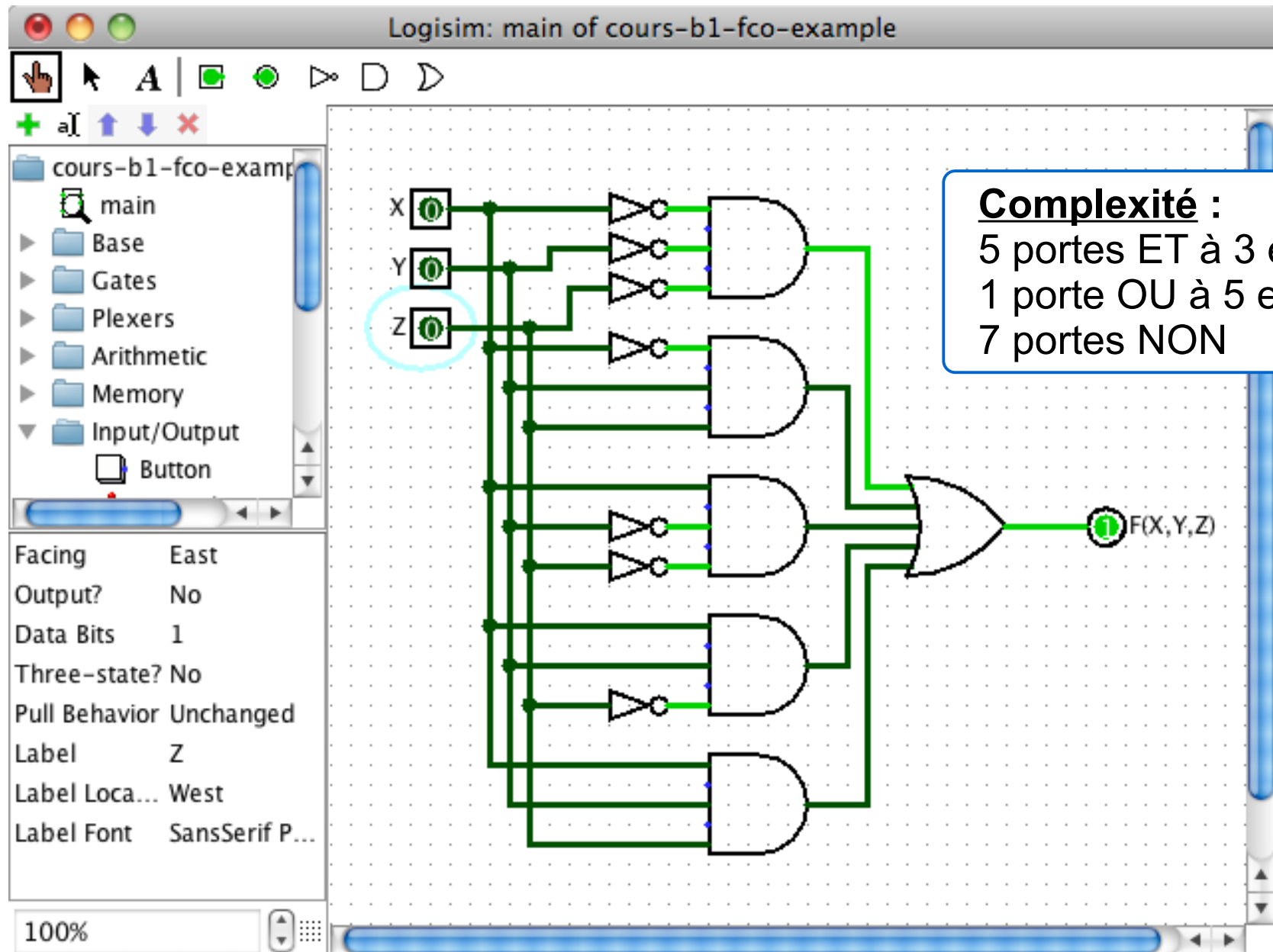
Optimalité

- L'expression donnée par la somme canonique est-elle optimale ?
- Nombre de *minterms* croît comme une fonction exponentielle du nombre de variables (2^N)
- Il existe des techniques visant à réduire la taille de l'expression logique et par conséquent le nombre de portes logiques requises et le nombre de leurs entrées.
 - des trucs (p. ex. tableaux de Karnaugh)
 - des algorithmes (p. ex. algorithme de Quine-McCluskey⁽¹⁾)
 - des outils implémentant certaines de ces techniques (p. ex. LogiSim⁽²⁾)

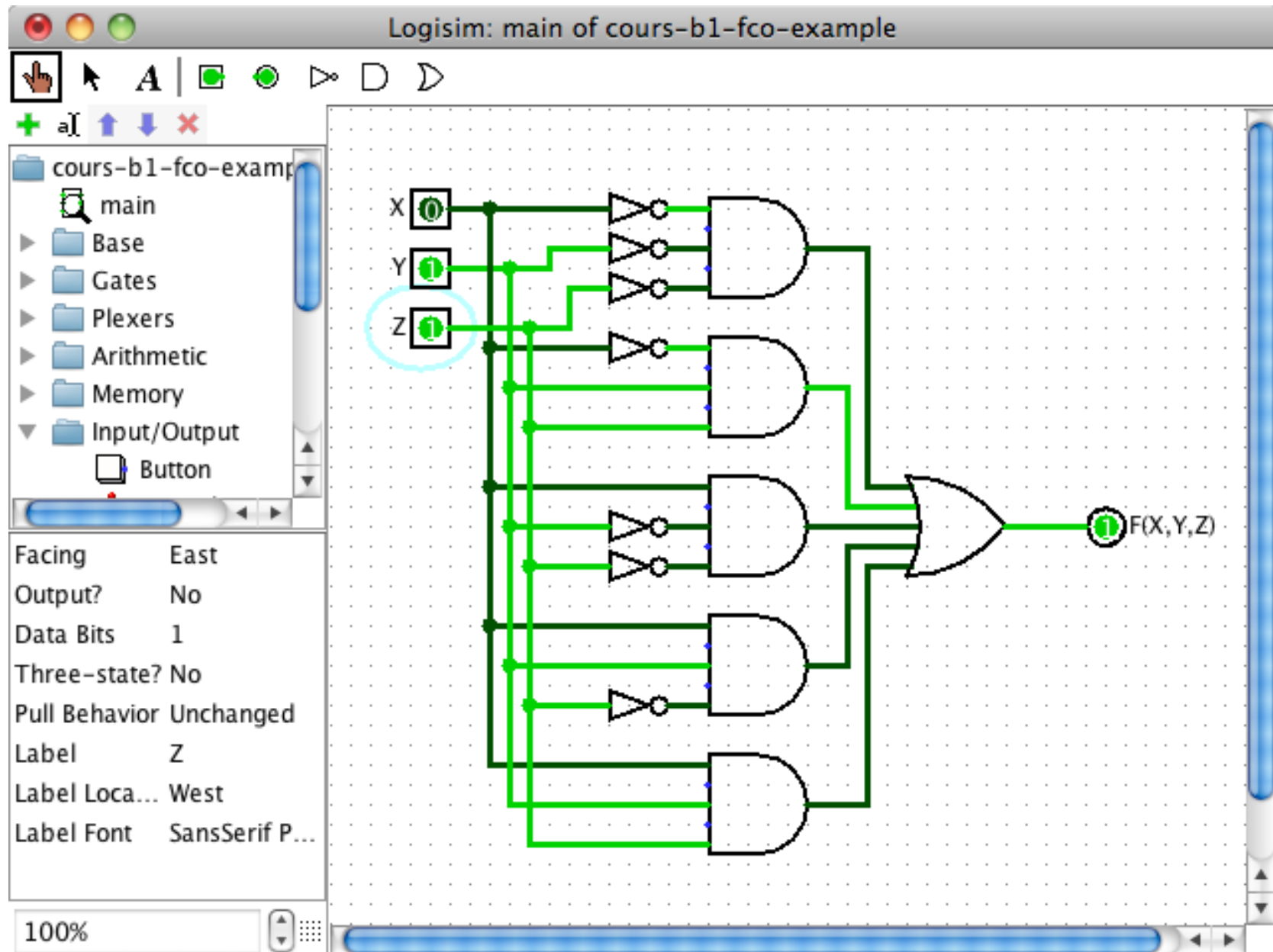
⁽¹⁾ Quine-McCluskey a une complexité élevée (exponentielle dans le nombre d'entrées) → utilisation d'heuristiques.

⁽²⁾ LogiSim est disponible gratuitement à l'adresse <http://www.cburch.com/logisim/index.html>

LogiSim



LogiSim



LogiSim

Expression logique

Output: FXYZ
 $\bar{X}\bar{Y}\bar{Z} + \bar{X}YZ + X\bar{Y}\bar{Z} + XY\bar{Z} + XYZ$

$\sim X \sim Y \sim Z + \sim X Y Z + X \sim Y \sim Z + X Y \sim Z + X Y Z$

Table de vérité

X	Y	Z	FXYZ
0	0	0	1
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

Minimisation
(et tableau de Karnaugh)

Output: FXYZ

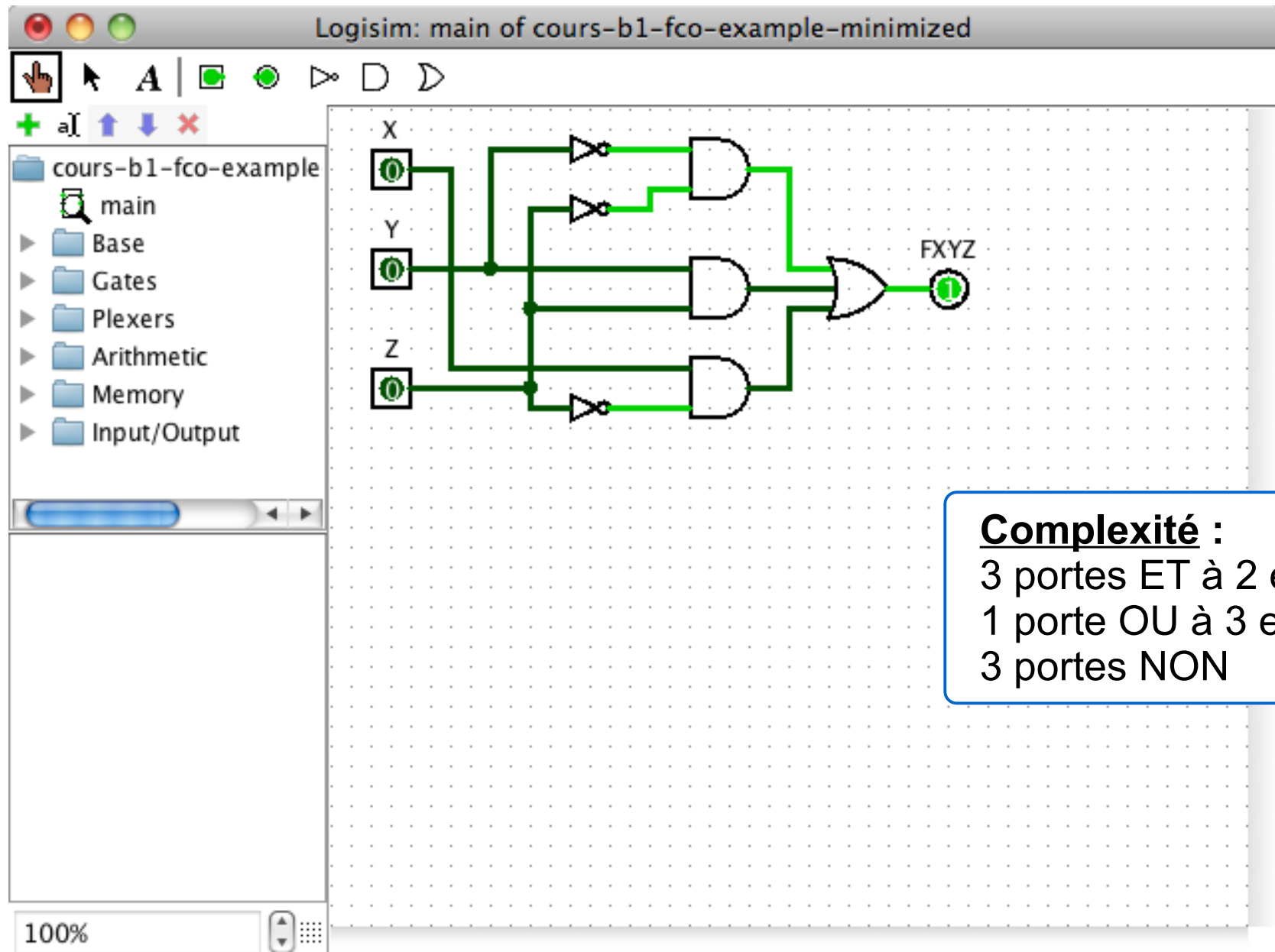
		Y, Z			
		00	01	11	10
X	0	1	0	1	0
	1	1	0	1	1

$\bar{Y}\bar{Z} + YZ + X\bar{Z}$

Set As Expression

Build Circuit

LogiSim



Complexité :

3 portes ET à 2 entrées
1 porte OU à 3 entrées
3 portes NON

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes

➡ Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Logique Combinatoire

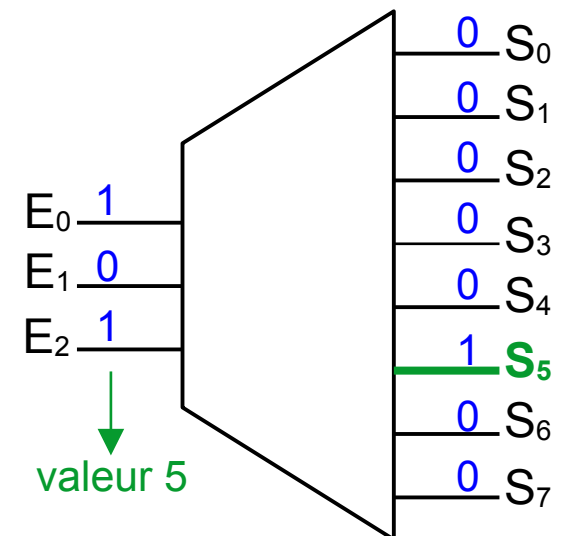
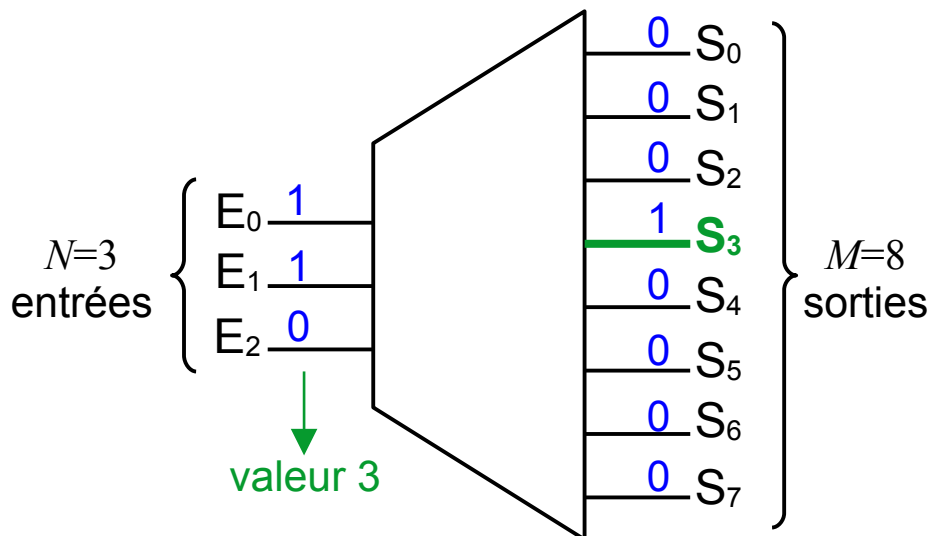
Blocs importants

- L'objectif de cette section est la description et la compréhension de blocs logiques réalisés en logique combinatoire.
- Les blocs logiques couverts par cette section sont
 - Décodeur binaire
 - Multiplexeur binaire
 - Demi-additionneur
 - Additionneur complet
- Ces blocs logiques nous serviront notamment à construire une unité arithmétique et logique (ALU) et la logique de contrôle d'un processeur complet...

Décodeur binaire

Principes

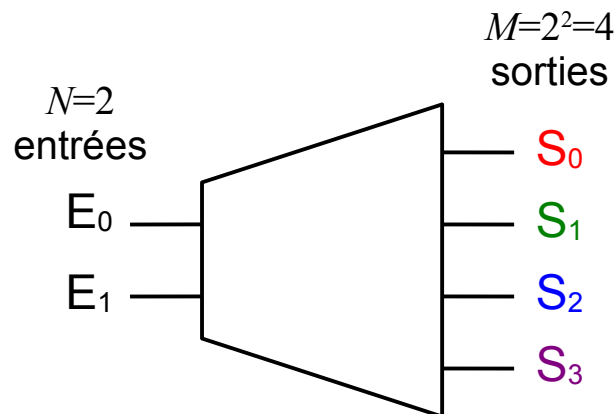
- Un décodeur binaire sert typiquement à choisir parmi plusieurs actions. Par exemple: l'opération qui doit être effectuée par une unité arithmétique et logique (ALU).
- Un **décodeur binaire** possède N entrées $E_i \in B$ et $M=2^N$ sorties $S_i \in B$. A tout moment, seule la sortie S_E est activée où E est le code présenté en entrée.
- Exemple : décodeur $N=3$ vers $M=8$



Décodeur binaire

Implémentation

- Le décodeur binaire peut être implémenté simplement en établissant sa table de vérité.
 - Le décodeur binaire est une fonction de $B^N \rightarrow B^M$. Chaque sortie correspond à une fonction qui n'est vraie que pour une combinaison unique des entrées $E_i \in B$. Un seul *minterm* suffit donc à exprimer la fonction qui correspond à chaque sortie.
- Exemple pour un décodeur 2 vers 4

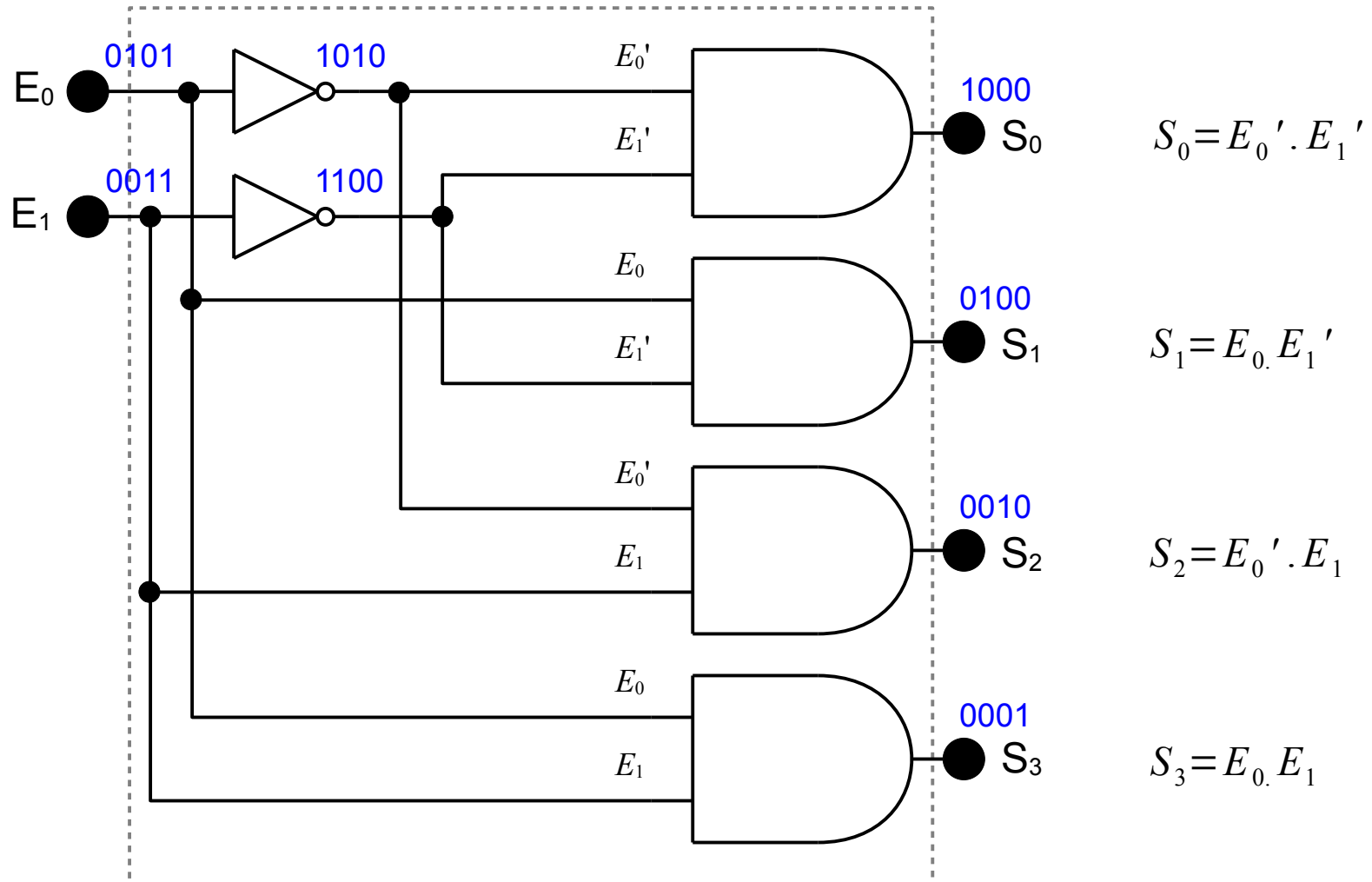


E_1	E_0	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$S_3 = E_0 \cdot E_1$
 $S_2 = E_0' \cdot E_1$
 $S_1 = E_0 \cdot E_1'$
 $S_0 = E_0' \cdot E_1'$

Décodeur binaire

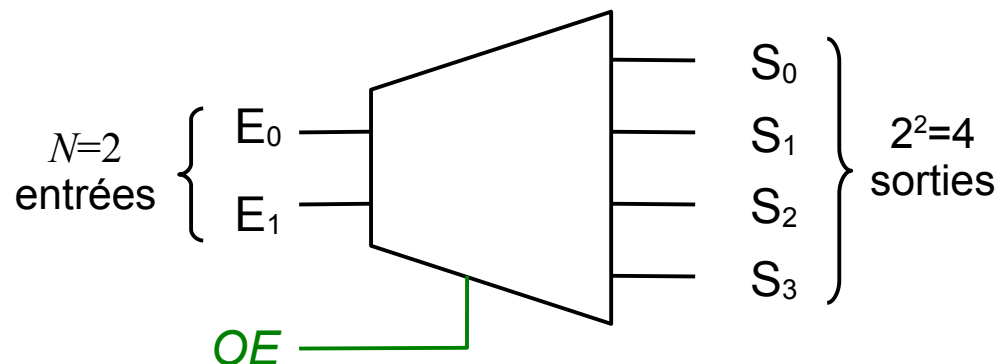
Implémentation



Décodeur binaire

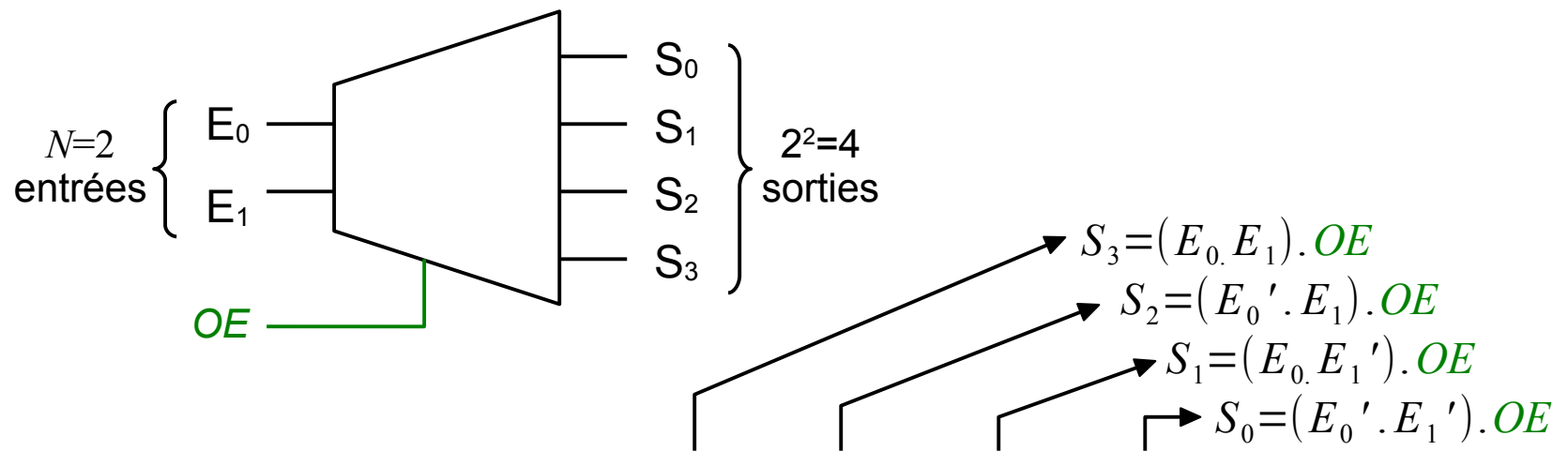
Variante - Décodeur binaire désactivable

- Un décodeur binaire possède parfois une **entrée supplémentaire appelée “output enable” (OE)** permettant de désactiver toutes les sorties.
 - Si $OE=0$, aucune sortie n'est activée.
 - Si $OE=1$, le décodeur fonctionne comme précédemment.
- On peut “voir” l'entrée OE comme faisant partie du code en entrée. Tous les codes dans lesquels $OE=0$ donnent le même résultat en sortie : toutes les sorties sont désactivées.
- Exemple : décodeur 2 vers 4 désactivable



Décodeur binaire

Décodeur binaire désactivable

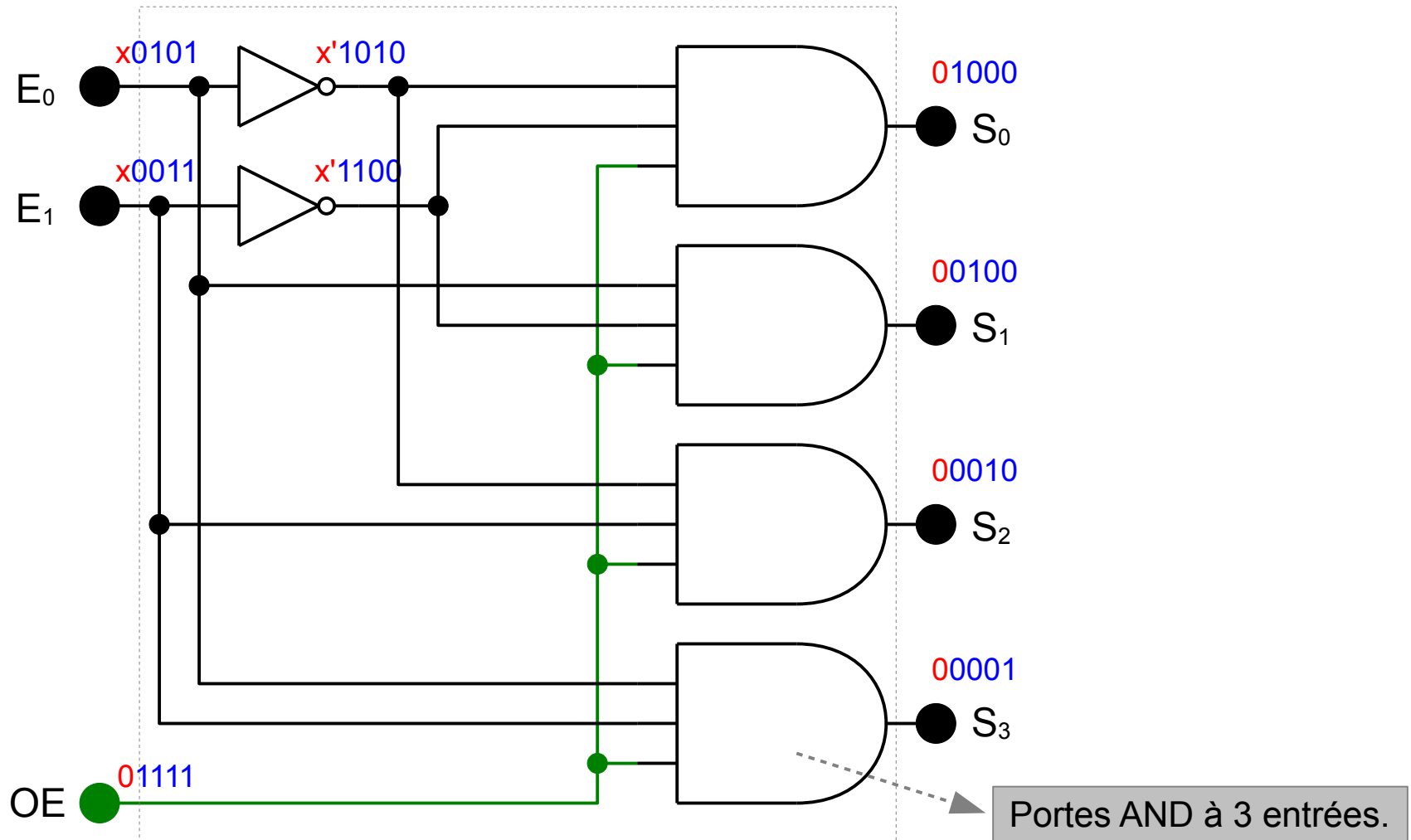


OE	E_1	E_0	S_3	S_2	S_1	S_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Les valeurs notées 'x' sont quelconques (0 ou 1).

Décodeur binaire

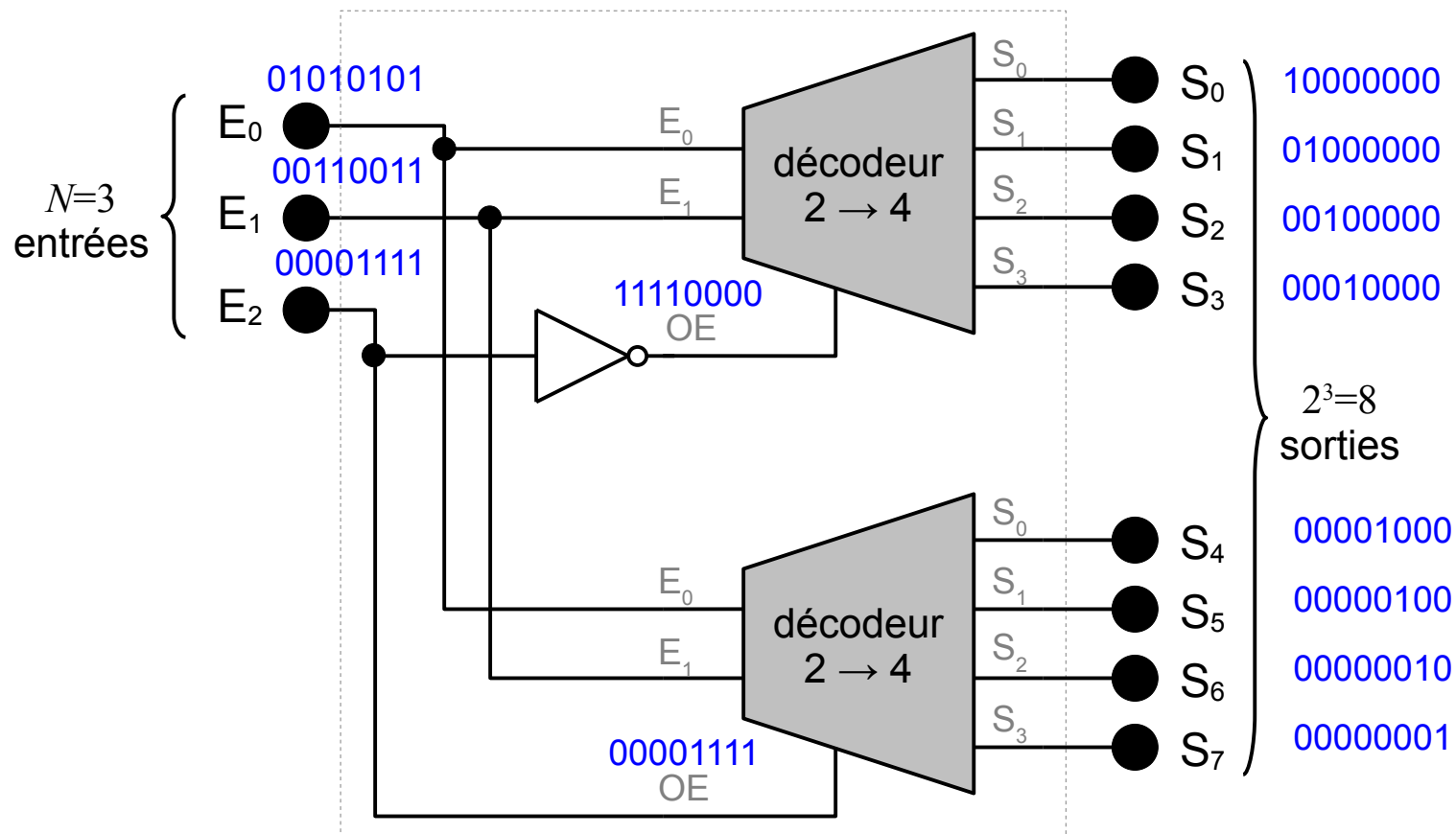
Décodeur binaire désactivable



Décodeur binaire

Cascade de décodeurs binaires

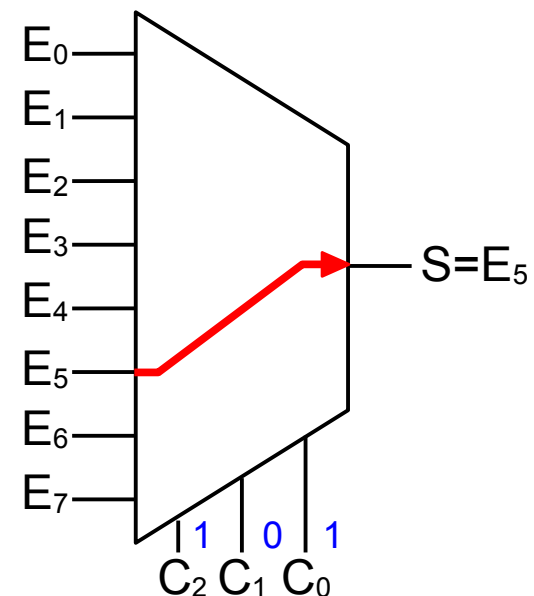
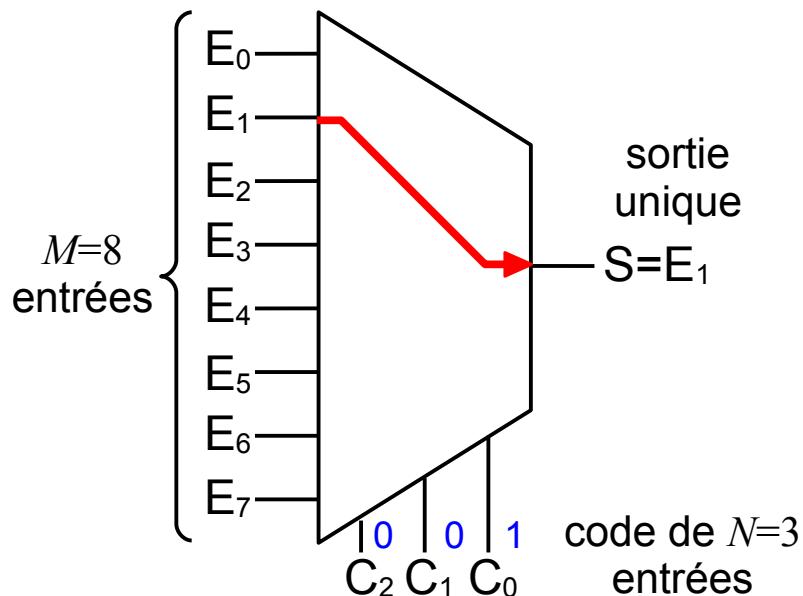
- Il est possible de construire un décodeur à N entrées à partir de décodeurs à $N-1$ entrées, avec $N > 2$.



Multiplexeur binaire

Principes

- Un multiplexeur binaire permet de sélectionner un signal binaire parmi plusieurs.
- Un **multiplexeur binaire** possède M entrées $E_i \in B$, un code C de N bits (avec $M=2^N$) ainsi qu'une unique sortie $S \in B$. La valeur de S est celle de l'entrée sélectionnée par C , autrement dit, $S=E_C$.
- Exemple : multiplexeur binaire à 8 entrées



Multiplexeur binaire

Implémentation

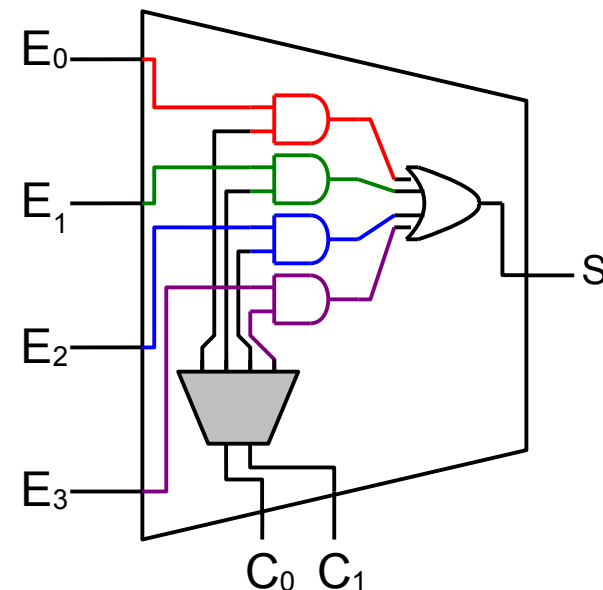
- Le multiplexeur binaire possède typiquement $N+2^N$ entrées et une sortie unique.
- Nous recherchons donc une fonction logique de $N+2^N$ variables.
- Le multiplexeur binaire a un fonctionnement similaire à un décodeur binaire dans le sens où le code de N bits sélectionne une des 2^N entrées. Cela correspond à associer à chaque entrée k un *minterm* unique sur les variables du code (noté $minterm_C(k)$)

- La sortie S prend la valeur de l'entrée k ssi le i lié est vrai.
Par exemple :

$$minterm_C(1) = C_0 \cdot C_1' \cdot E_1$$

- Au final :

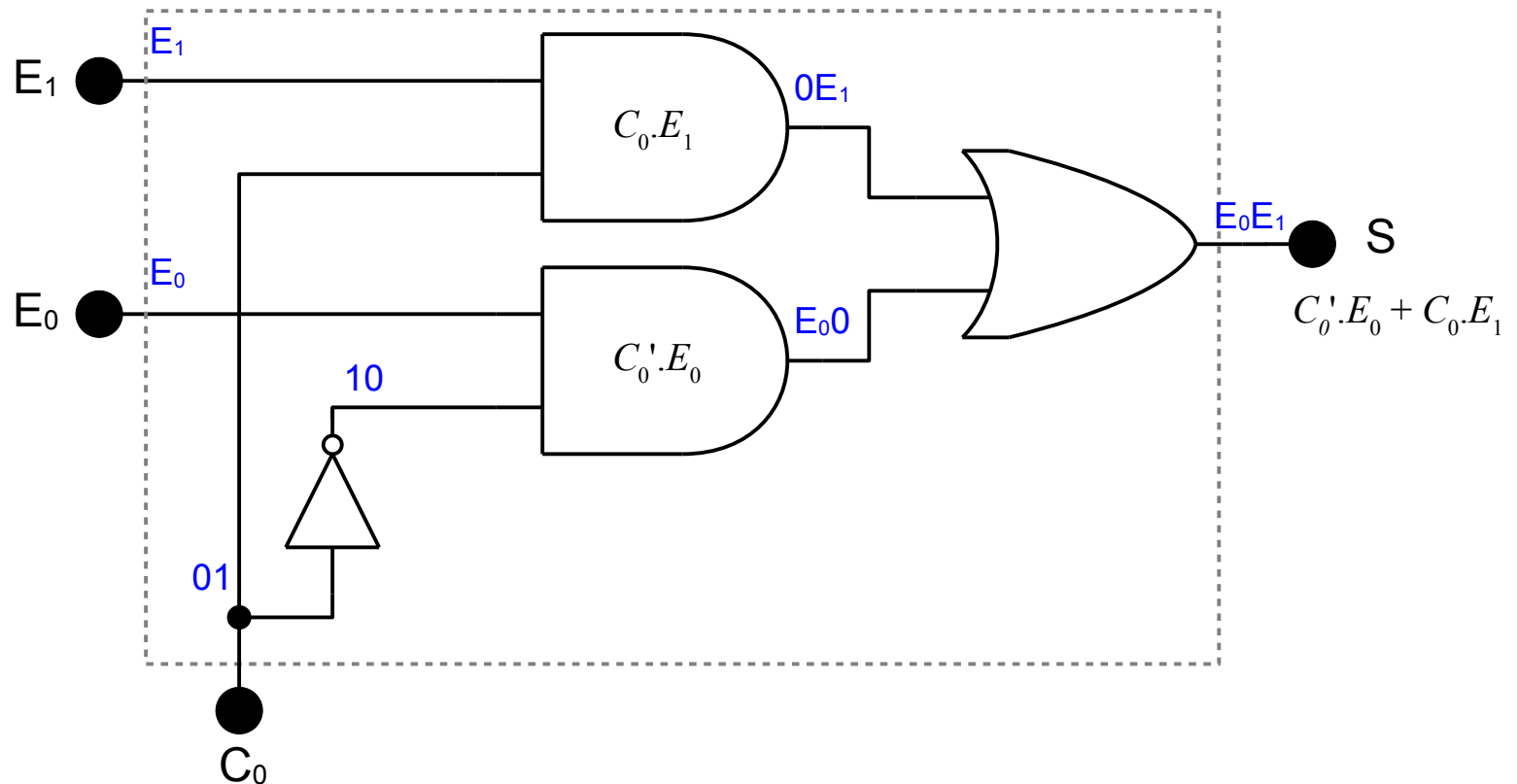
$$S = \sum_{i=0}^{M-1} minterm_C(i) \cdot E_i$$



Multiplexeur binaire

Implémentation

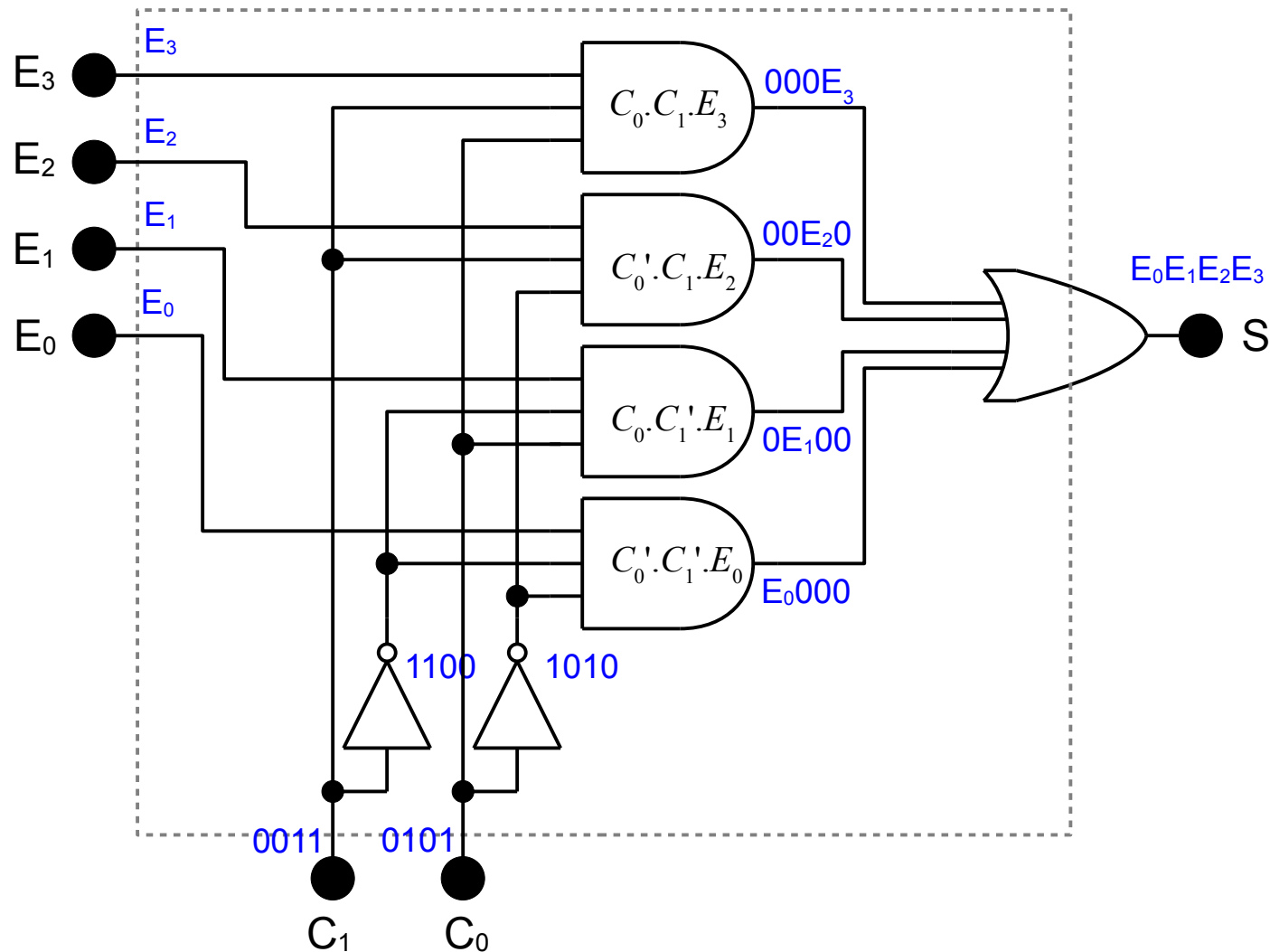
- Exemple : multiplexeur binaire à 2 entrées



Multiplexeur binaire

Implémentation

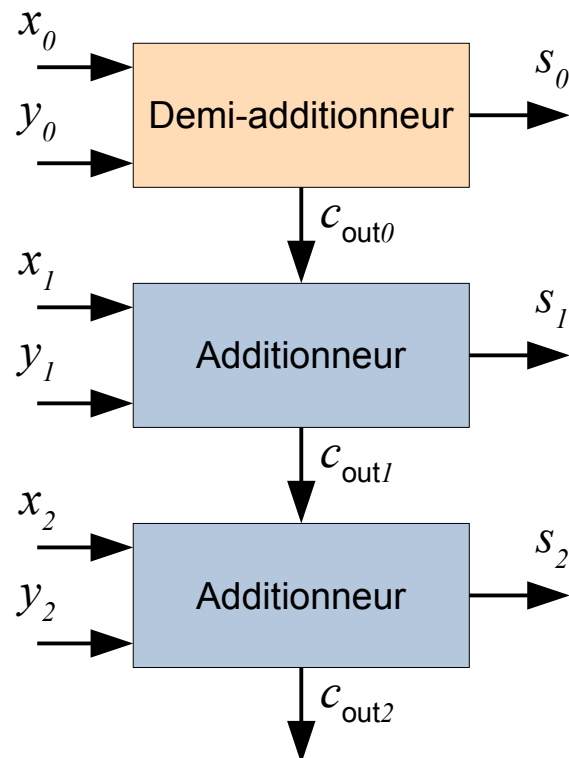
- Exemple : multiplexeur binaire à 2 entrées



Additionneur

Additionneur à propagation de report

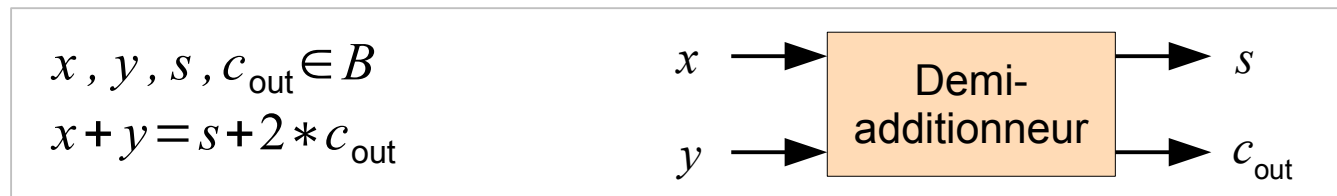
- L'addition peut être réalisée avec une cascade d'additionneurs (*ripple-carry-adder*). Le premier additionneur ne prend pas de report entrant et est appelé demi-additionneur.



Additionneur

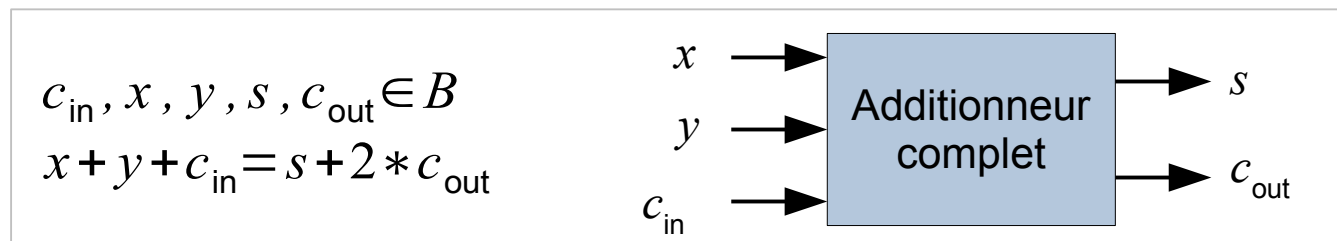
Demi-additionneur

- Le demi-additionneur permet d'additionner 2 bits (x et y) et de déterminer la somme s (modulo 2) et un report c_{out} .
- Il s'agit d'une fonction logique de B^2 vers B^2 .



Additionneur (complet)

- L'additionneur complet effectue l'addition de 2 bits (x et y) et d'un report entrant c_{in} . Il produit en sortie une somme s (modulo 2) et un report sortant c_{out} .
- Il s'agit d'une fonction logique de B^3 vers B^2 .



Additionneur

Implémentation – Demi-additionneur

- Afin d'implémenter le demi-additionneur, il est aisé d'établir sa table de vérité, puis d'en dériver une expression logique.
- Remarquons que
 - s est obtenu en effectuant un OU exclusif (XOR) entre x et y ,
 - c_{out} n'est vrai que si x et y sont vrais simultanément (ET).

x	y	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s = x \oplus y$$

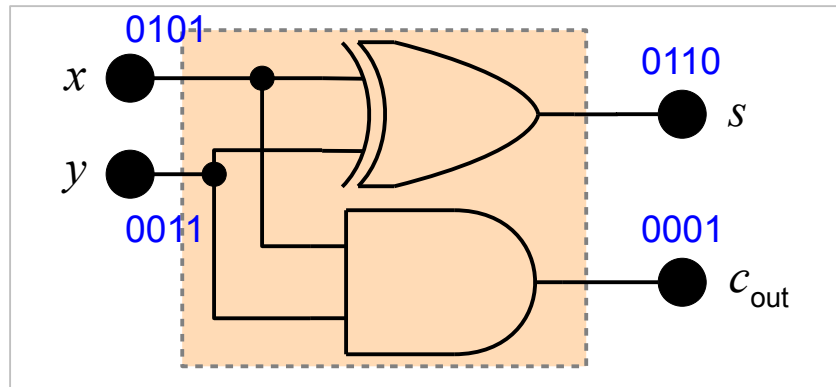
$$c_{out} = x \cdot y$$

- Note : il est également possible de dériver ces fonctions logiques en utilisant les *minterms* ou les *maxterms*.

Additionneur

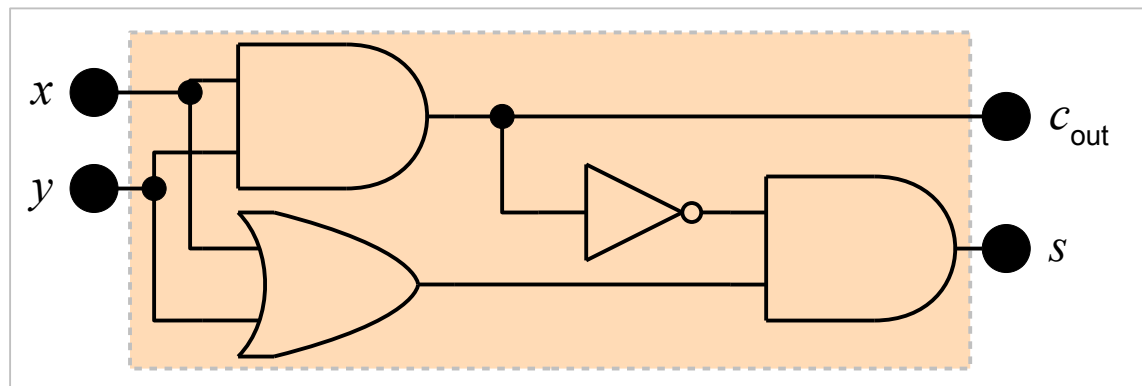
Implémentation – Demi-additionneur

- Les expressions logiques pour s et c_{out} donnent lieu au circuit logique suivant.



$$s = x \oplus y$$
$$c_{out} = x \cdot y$$

- En montrant l'implémentation interne de la fonction XOR, on constate que le calcul du *carry-out* est déjà présent.



Additionneur

Implémentation – Additionneur complet

- L'implémentation de l'additionneur complet peut également être dérivée à partir de la table de vérité. Une approche possible consiste à séparer les cas où le report entrant c_{in} vaut 1 des cas où il vaut 0.

c_{in}	x	y	s	c_{out}
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

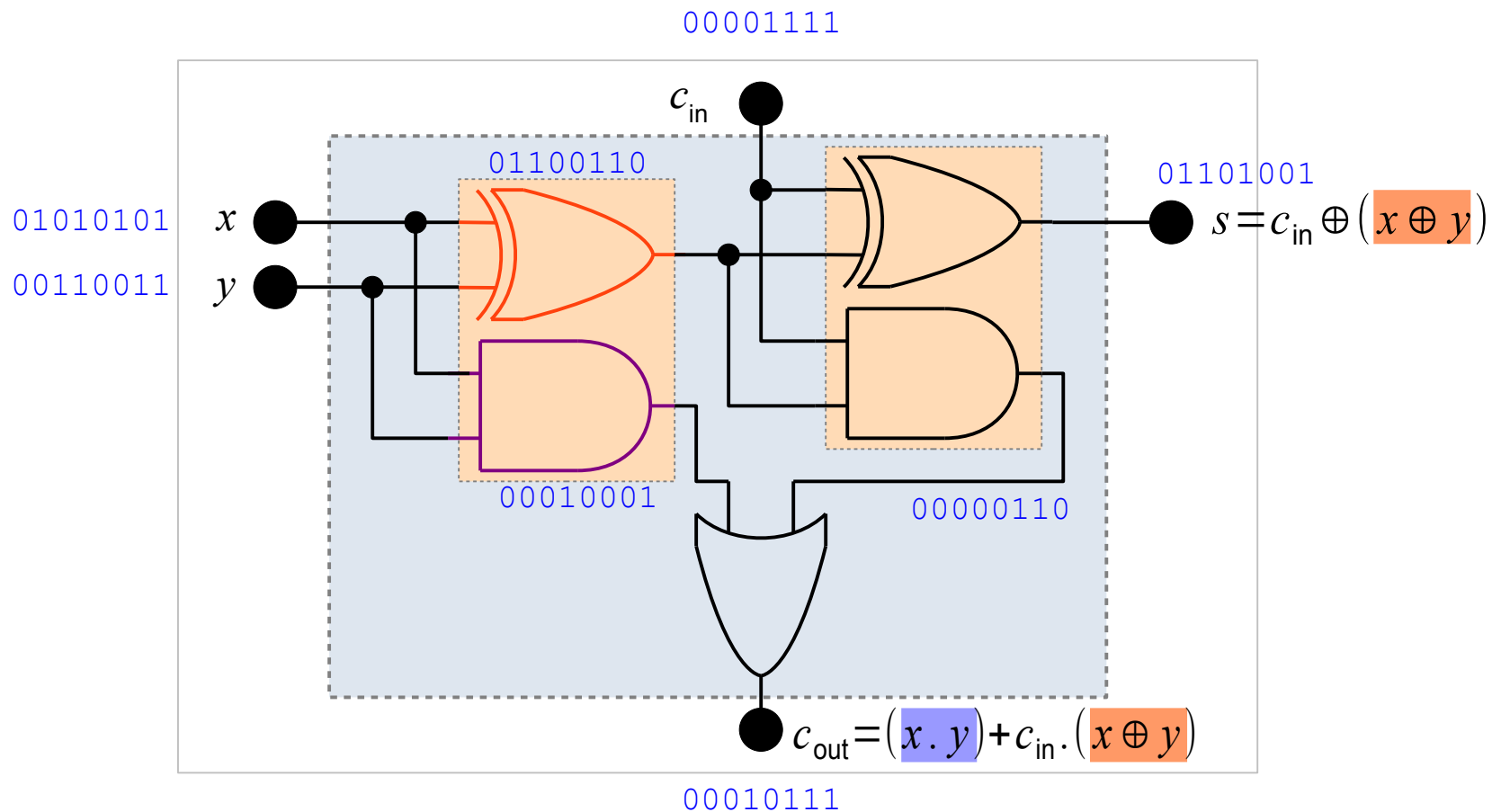
$$s = (c_{in}' \cdot (x \oplus y)) + (c_{in} \cdot (x \oplus y)') = c_{in} \oplus x \oplus y$$

$$c_{out} = (x \cdot y) + c_{in} \cdot (x \oplus y)$$

Additionneur

Implémentation – Additionneur complet

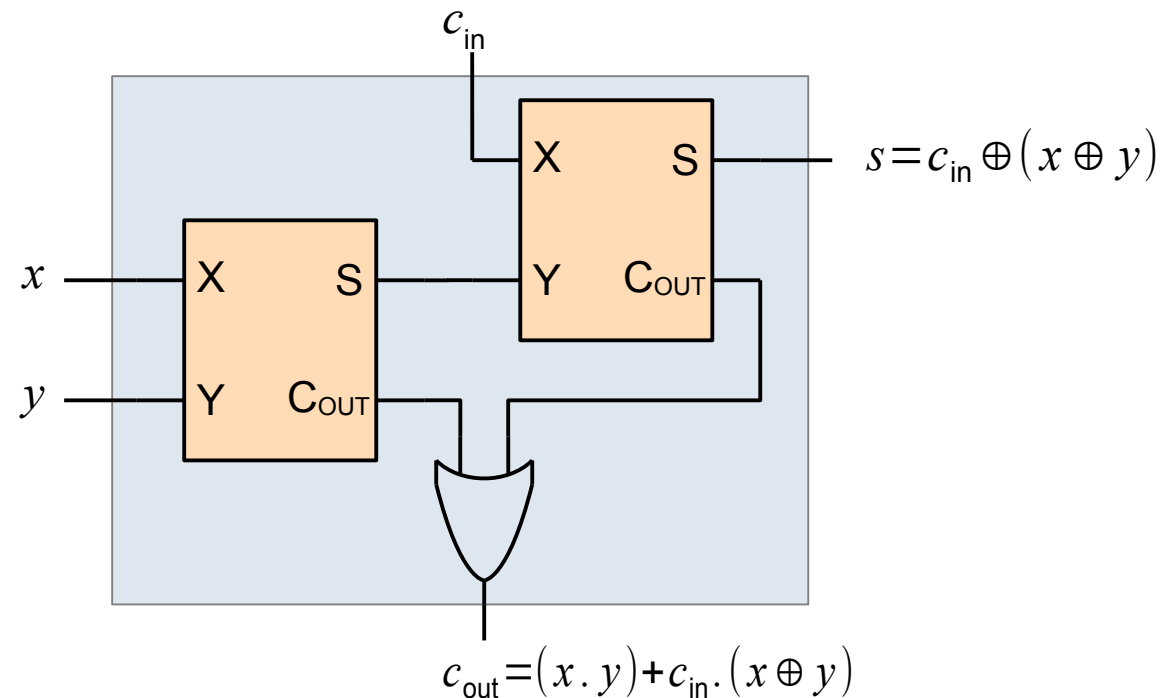
- Les expressions logiques mènent au circuit logique suivant.



Additionneur

Implémentation – Additionneur complet

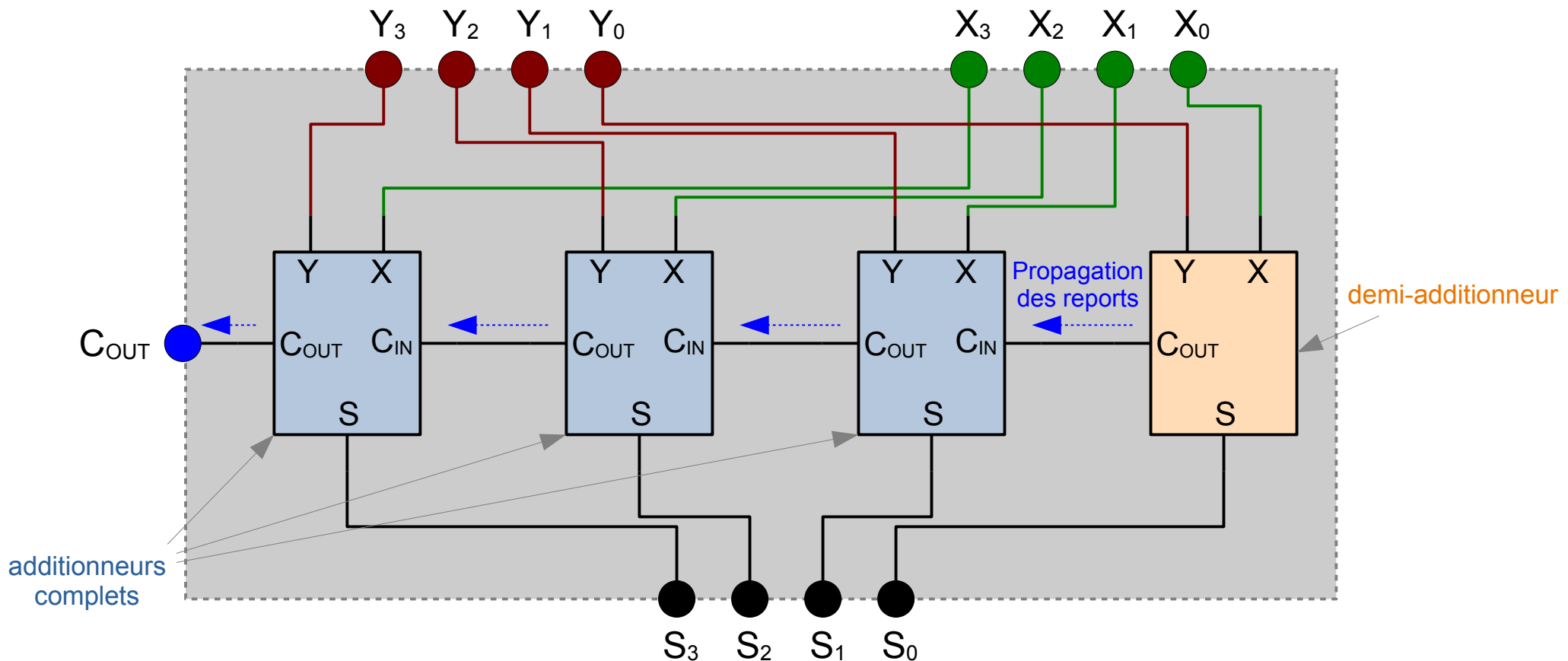
- L'additionneur complet peut également être obtenu en utilisant deux demi-additionneurs.



Additionneur

Additionneur N -bits

- L'additionneur N -bits (*ripple-carry adder*) additionne deux nombres de N bits. Il donne également 1 bit de report qui permet de déterminer s'il y a dépassement.



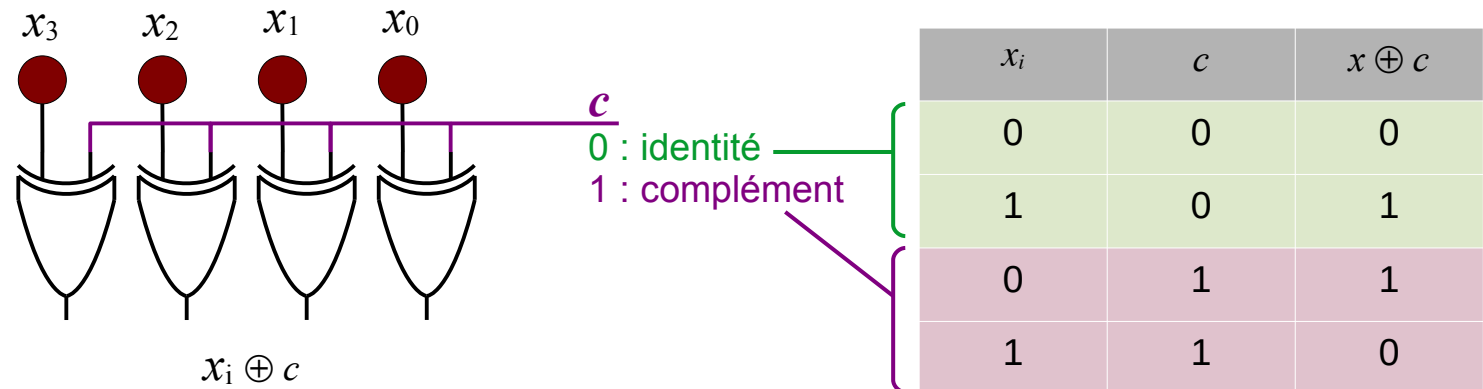
Additionneur

Soustraction N -bits

- La soustraction de x et y est transformée en addition de x avec l'opposé de y (rappel Chapitre 2). L'opposé de y est calculé en **complémentant** tous les bits de y , puis en **ajoutant 1** au nombre obtenu (report entrant initial de 1).

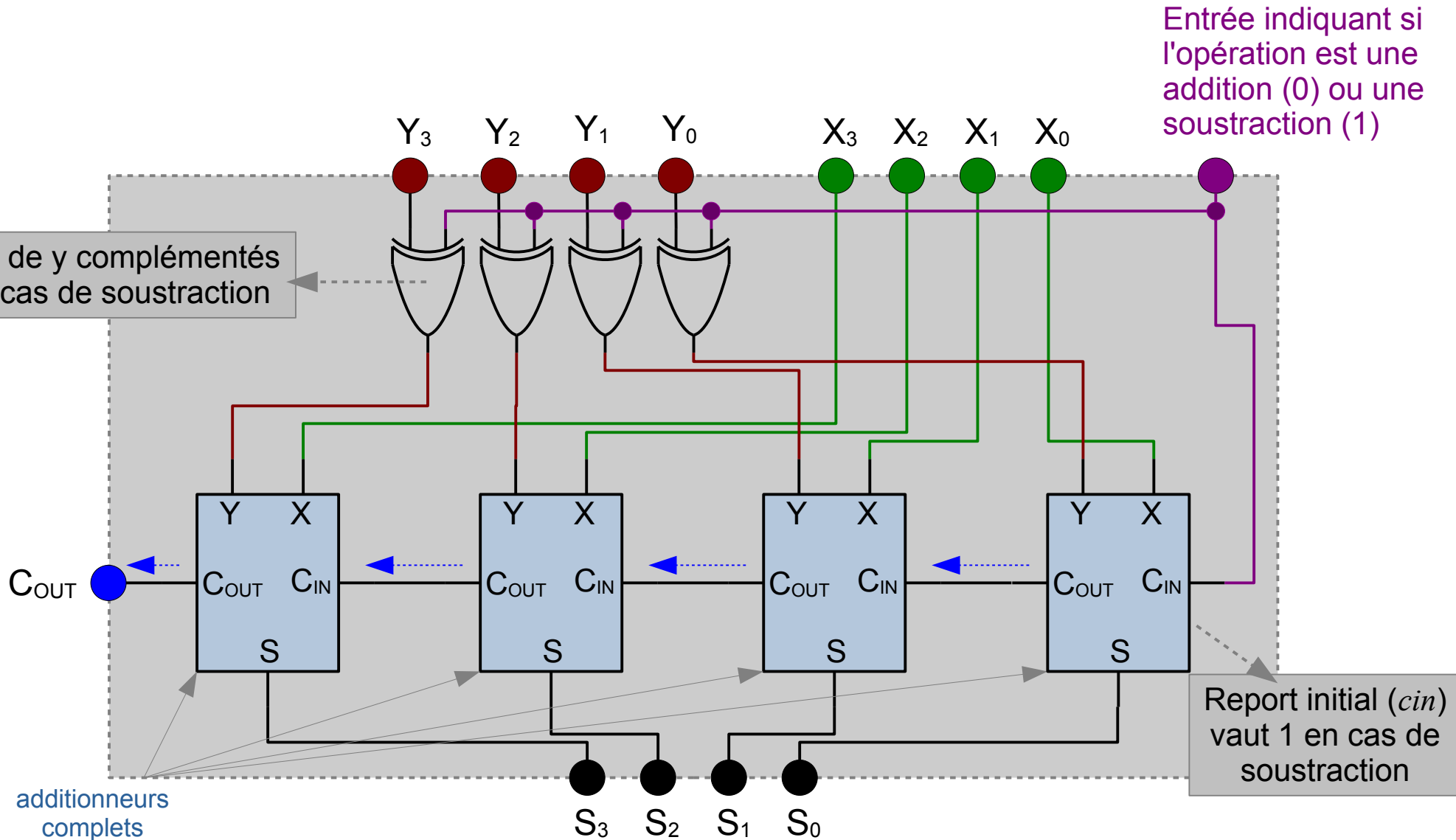
$$\begin{aligned}x - y &= x + (-y) \\&= x + ((2^N - 1) - y + 1) \\&= x + (\underbrace{(2^N - 1) - y}_{\text{Complémenter les bits de } y}) + \underbrace{1}_{\text{Report initial } (c_{in}) \text{ vaut } 1}\end{aligned}$$

- Pour complémenter les bits de y en cas de soustraction, des portes XOR peuvent être utilisées. Le circuit logique suivant permet d'agir de manière contrôlable comme **identité** ou **complément** sur un mot de N bits



Additionneur

Implémentation – Addition / soustraction N -bits



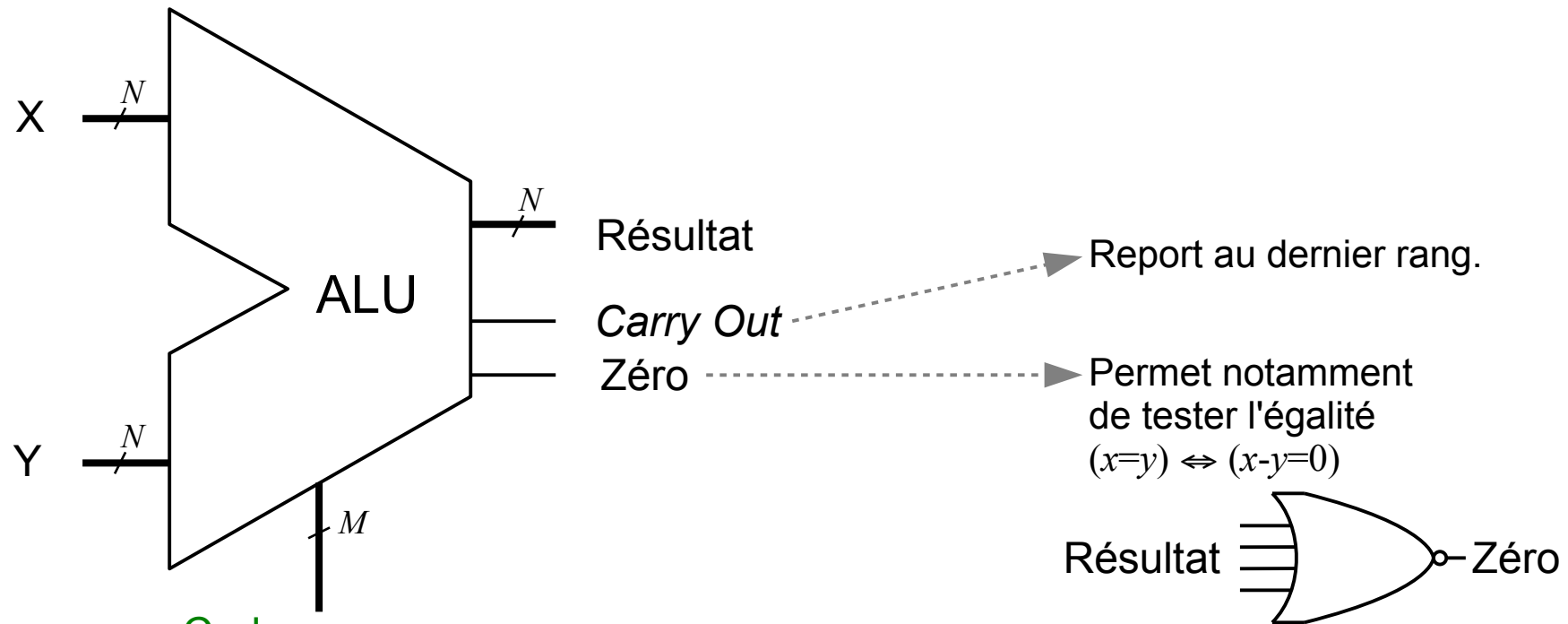
Unité Arithmétique et Logique (ALU)

Principes

- L'**unité arithmétique et logique** (*arithmetic and logic unit* – **ALU**) est un bloc de base important dans un processeur. L'ALU est capable d'effectuer des opérations arithmétiques diverses sur une ou plusieurs (typiquement 2) données présentées sur ces entrées.
- Les opérations suivantes sont généralement supportées
 - addition, soustraction, multiplication, division, opposé
 - comparaisons: égalité, plus grand que, strictement plus grand que, plus petit que, strictement plus petit que
 - décalages à gauche / à droite, rotations à gauche / à droite
 - valeurs constantes (p.ex. 0)
 - opérations bit à bit: ET, OU, NON (complément), OU-exclusif
- L'opération arithmétique effectuée est sélectionnée par un **code d'opération** présenté sur un autre ensemble d'entrées.

Unité Arithmétique et Logique (ALU)

Représentation standard



Code
opération
ALU

Code	Fonct.	Résultat
0	ADD	$X+Y$
1	SUB	$X-Y$
2	AND	$X \text{ ET } Y$
3	OR	$X \text{ OU } Y$
4	NOT	NON X
5	LESS	1 si $(X < Y)$; 0 sinon
...

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

Principes

- Verrou, Bascule bistable, Registre
- Machines à états
- Mémoire

Logique Séquentielle

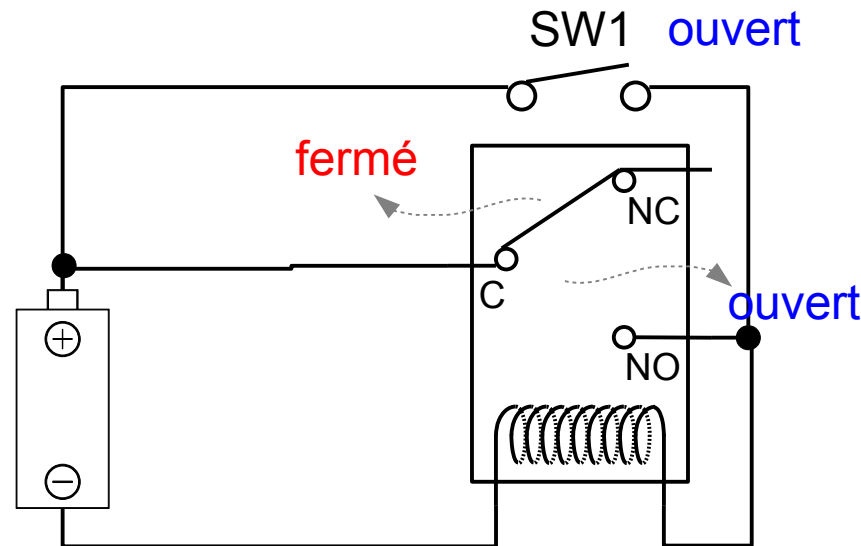
Introduction

- En logique combinatoire, les valeurs des sorties d'un circuit ne dépendent que des valeurs des entrées. Pour cela, il ne doit pas y avoir de boucle dans le circuit.
- En **logique séquentielle** (*sequential logic*), les sorties dépendent des entrées ainsi que de l'historique du circuit (les valeurs précédentes des entrées). Les circuits séquentiels contiennent généralement des **boucles de rétroaction** (*feedback loop*) qui permettent de conserver un état.
- La logique séquentielle permet d'implémenter des **machine à états** (*state-machines*).

Logique Séquentielle

Introduction

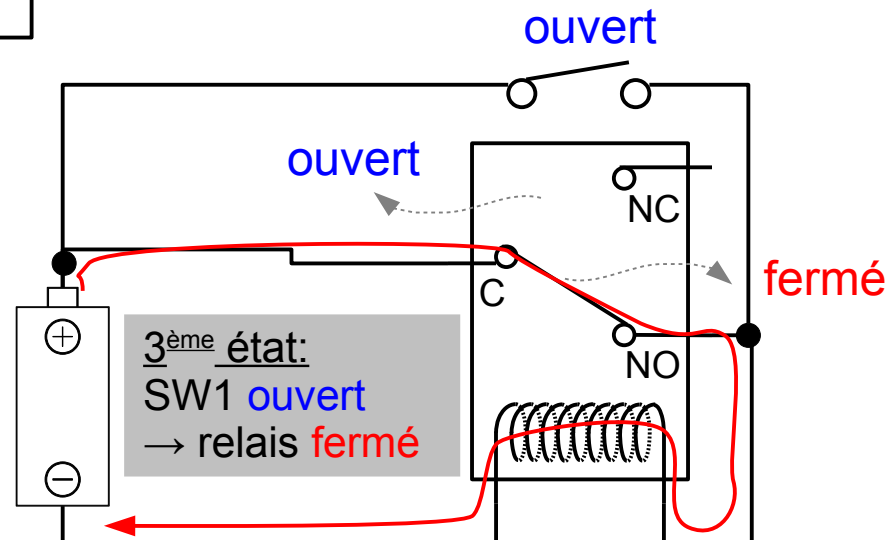
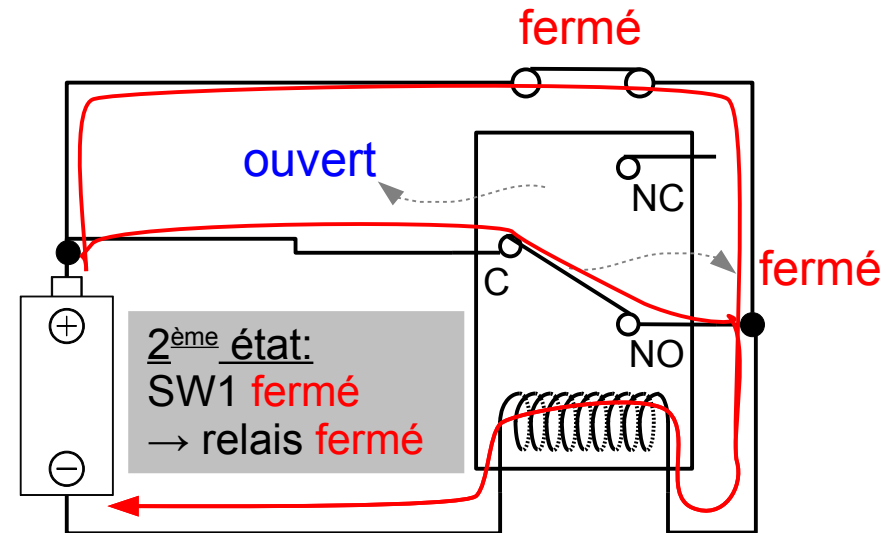
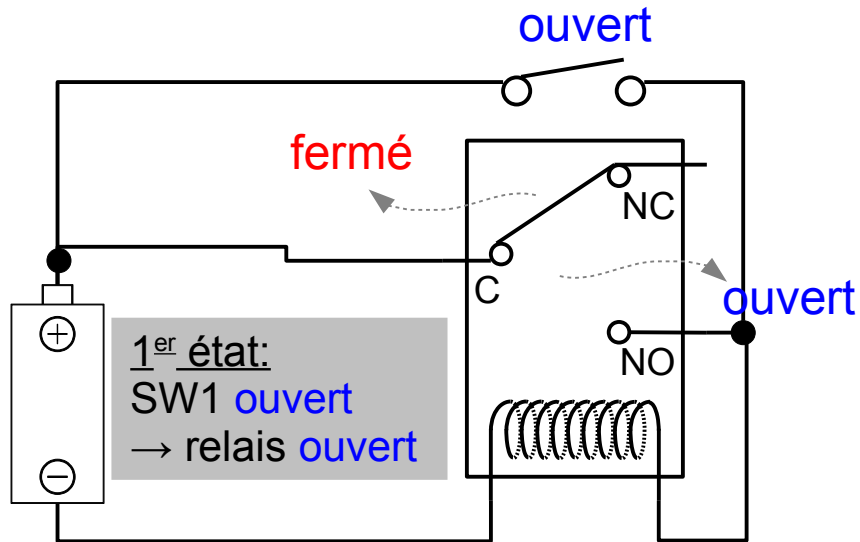
- Revenons à nos circuits électriques à relais. Quel est le comportement du circuit ci-dessous ?



- Le relais a un impact sur le circuit qui le commande. En effet, le contact normalement ouvert du relais est en parallèle avec SW1 ! Il s'agit d'une boucle de rétroaction.

Logique Séquentielle

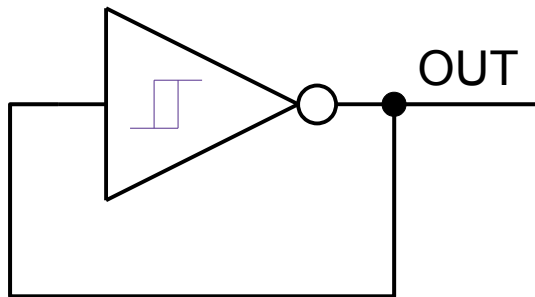
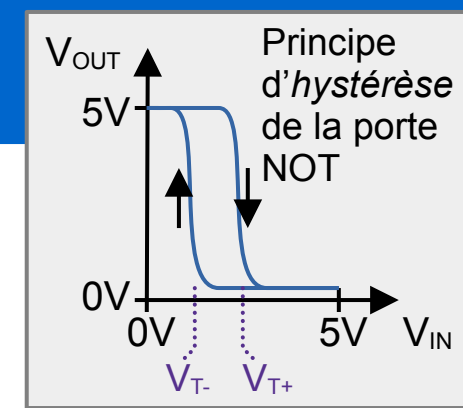
Introduction



Logique Séquentielle

Expérience – Oscillateur

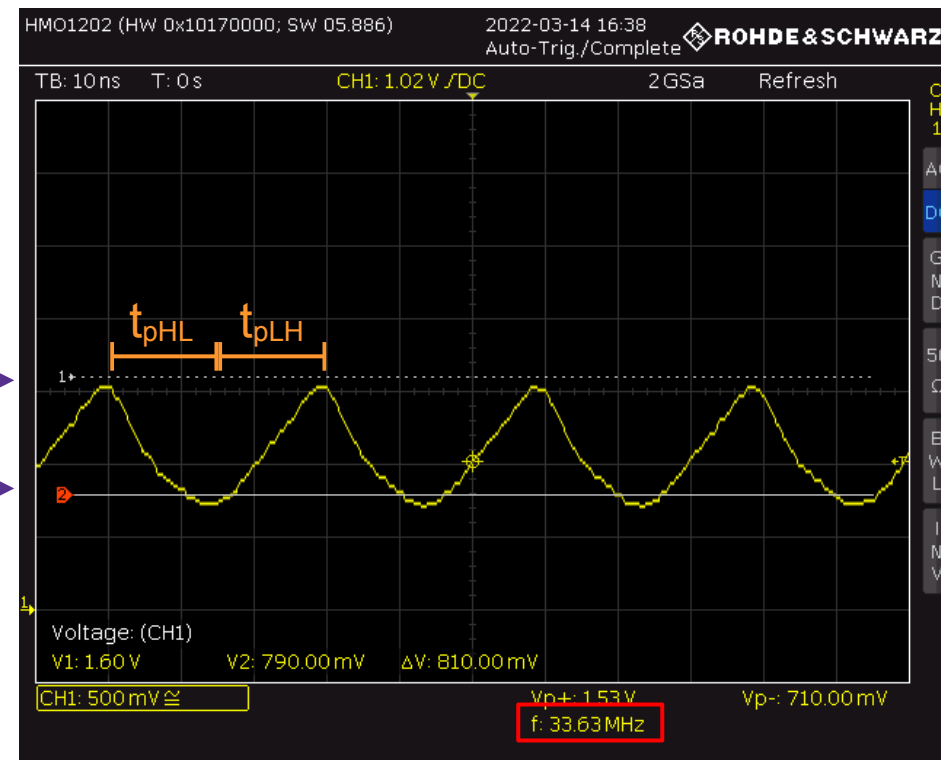
- Circuit intégré 74LS14 – hex *schmitt-trigger* NOT gates



- Seuils de transition (*typ.*)
 - $V_{T+} = 1.6 \text{ V}$
 - $V_{T-} = 0,8 \text{ V}$
- Délais de propagation (*typ.*)
 - $t_{pLH} = 15 \text{ ns}$
 - $t_{pHL} = 15 \text{ ns}$

$V_{T+} \rightarrow$

$V_{T-} \rightarrow$



$$f = \frac{1}{t_{pLH} + t_{pHL}} \approx 33 \text{ MHz}$$

Verrous et bascules bistables

- Les **verrous** (*latches*) et les **bascules bistables** (*flip-flops*) sont les éléments de base de la plupart des circuits en logique séquentielle.
- Les objectifs des verrous et des bascules bistables sont très similaires: **permettre la conservation d'un état logique (1/0)**. La façon dont les verrous et les bascules bistables sont utilisés et leurs fonctionnements sont différents.
- Il existe différentes versions des verrous et des bascules bistables. Ce cours n'en introduit que quelques-unes.

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

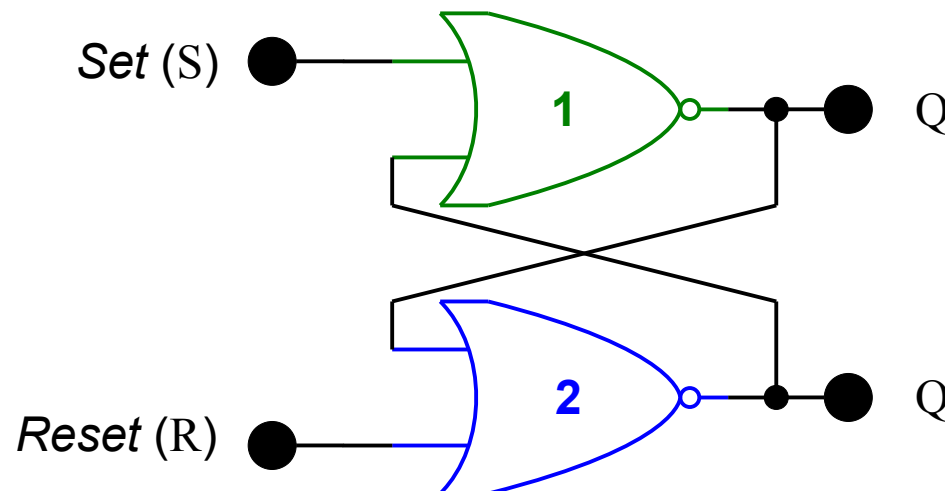
Logique séquentielle

- Principes
- ➡ Verrou, Bascule bistable, Registre
- Machines à états
 - Mémoire

Logique Séquentielle

Verrou S-R

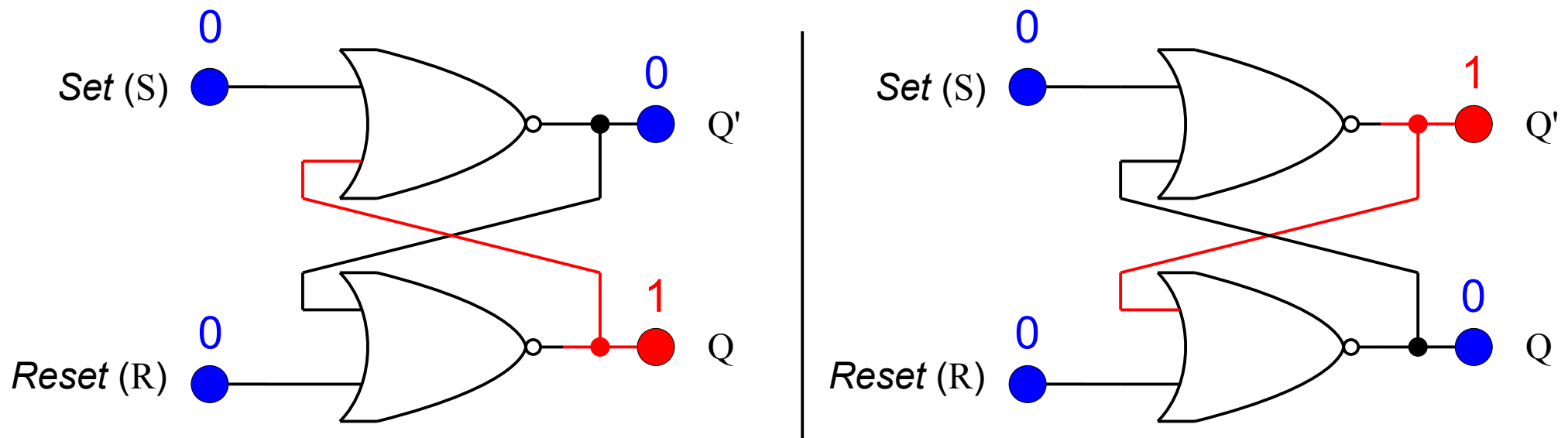
- Le **verrou S-R** (*SR-latch*) a pour objectif de maintenir un état. Il s'agit donc d'une implémentation possible pour une mémoire à 1 bit.
- Le verrou S-R repose sur l'utilisation de 2 portes NON-OU
- Ce qui le distingue d'un “banal” circuit combinatoire c'est qu'il contient une boucle :
 - sortie de porte NON-OU 1 → entrée de porte NON-OU 2
 - sortie de porte NON-OU 2 → entrée de porte NON-OU 1



Logique Séquentielle

Verrou S-R

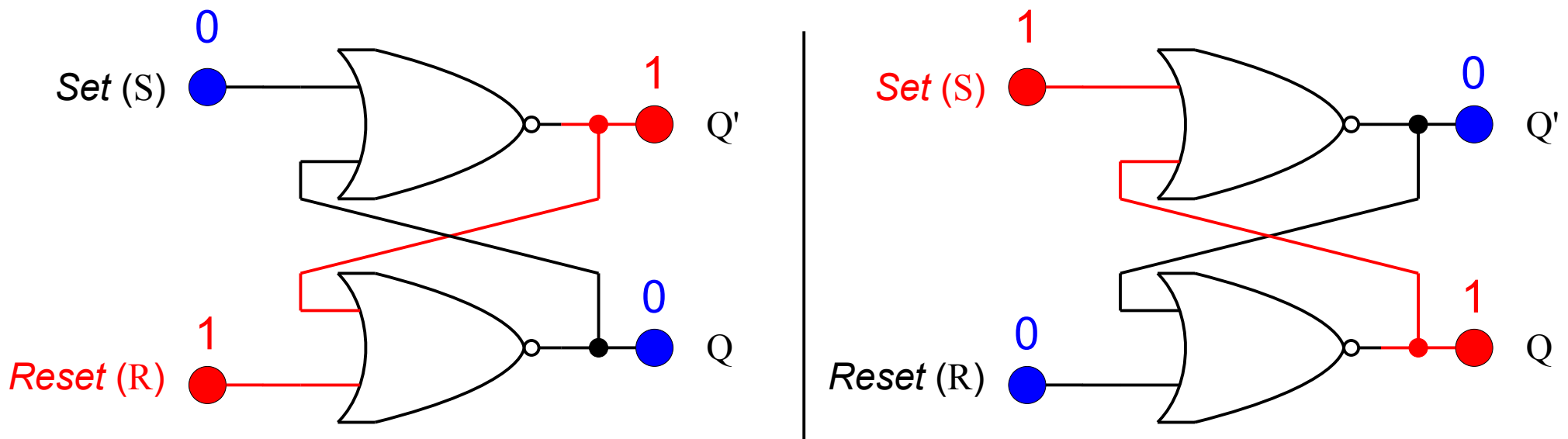
- Le verrou possède **deux états stables**. Ceux-ci sont atteints lorsque les entrées S et R sont toutes les deux inactives (i.e. elles valent 0). Ces deux états sont montrés ci-dessous:
 - Etat de gauche : sortie Q à 1 (et complément Q' à 0)
 - Etat de droite : sortie Q à 0 (et complément Q' à 1)



Logique Séquentielle

Verrou S-R

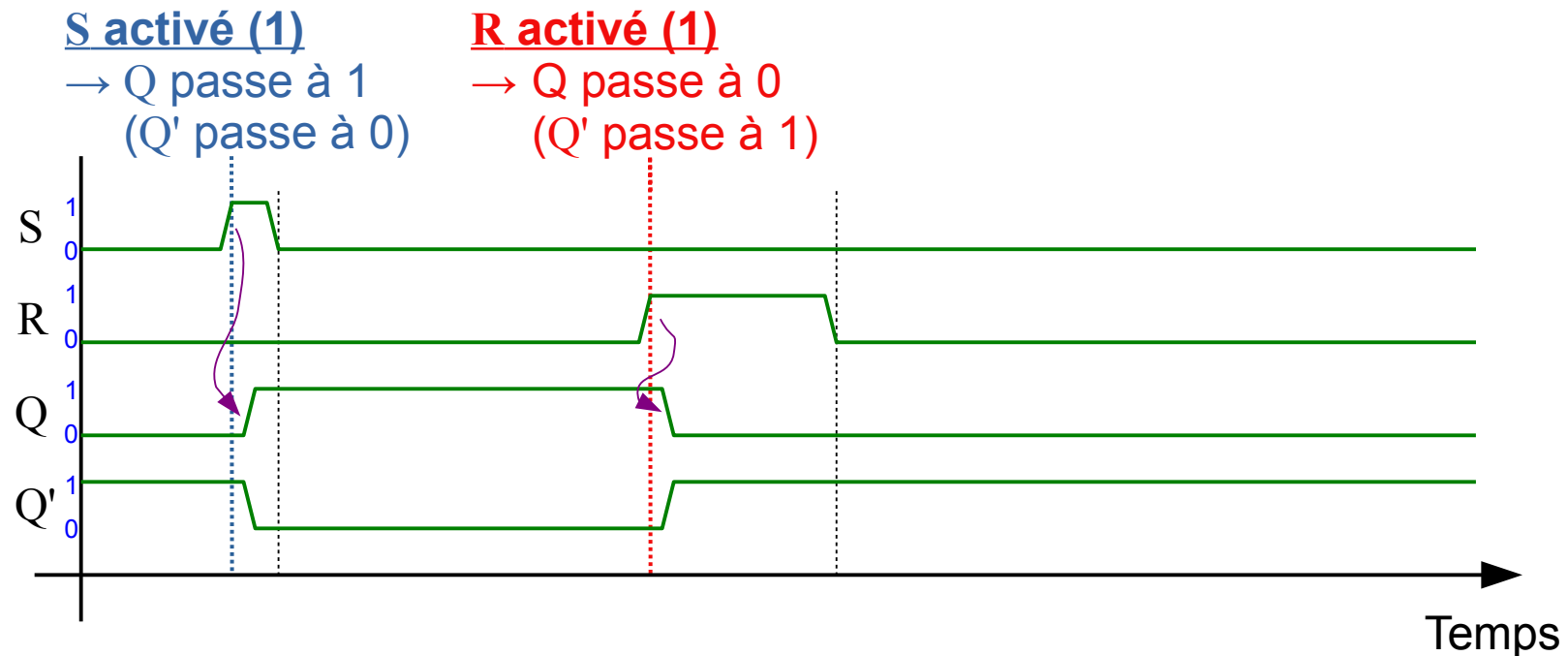
- Les deux entrées S et R permettent de forcer un changement de l'état du verrou, i.e. une **transition**, en imposant une valeur dans la boucle de rétroaction.
 - L'entrée S (*Set*) force la sortie Q à 1
 - L'entrée R (*Reset*) force la sortie Q à 0



Logique Séquentielle

Verrou S-R

- Le diagramme suivant illustre le fonctionnement d'un verrou S-R.

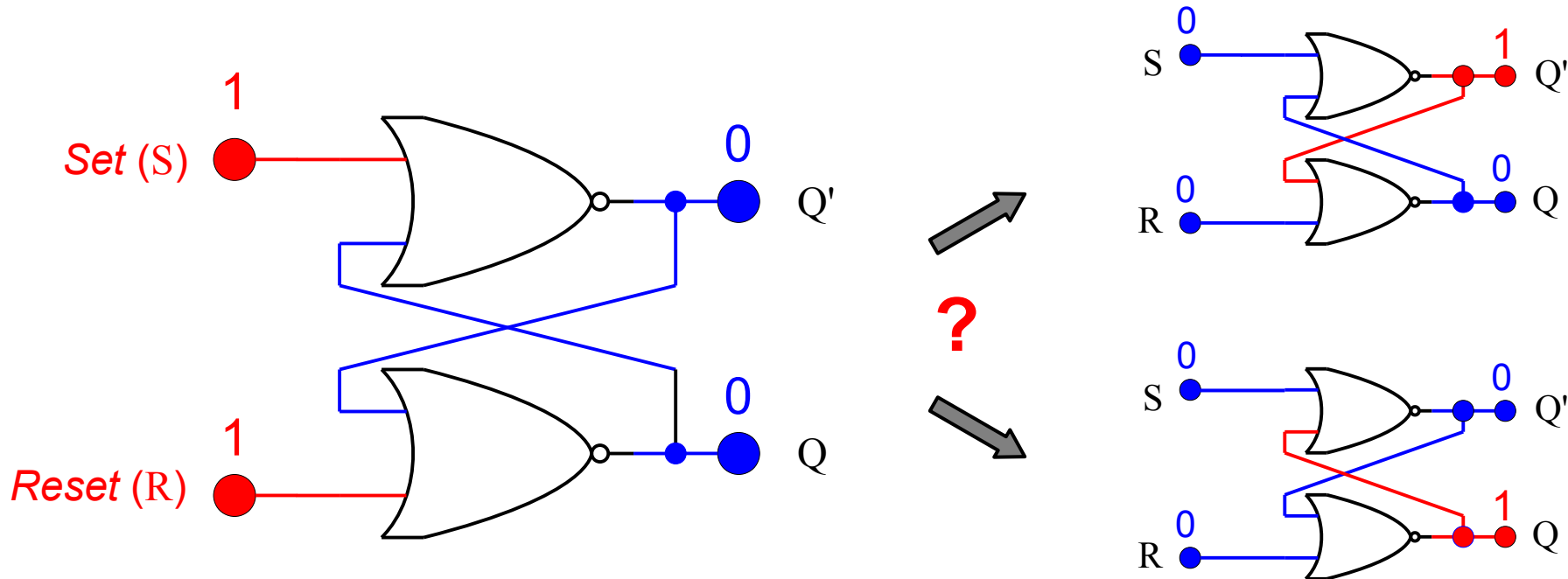


- Note : le **décalage** entre le moment où S (resp. R) passe à 1 et celui où Q passe à 1 change (resp. Q passe à 0) illustre le délai de réaction du verrou.

Logique Séquentielle

Méta-stabilité

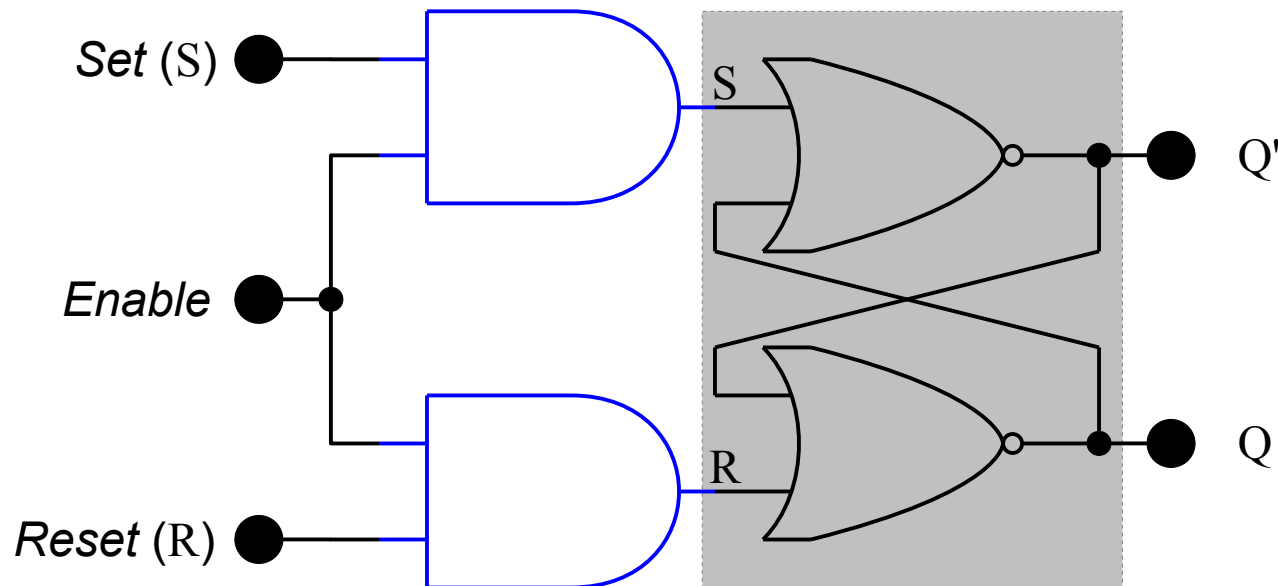
- Que se passe-t-il si les entrées S et R sont activées simultanément ?
- Le verrou entre alors dans un état contradictoire. Les deux sorties Q et Q' qui devraient être complémentaires valent toutes les deux 0. Cet état est appelé **méta-stable**.
- Lorsque S et R sont désactivés (0) simultanément, le verrou passe dans un état **indéfini** ! (non prévisible)



Logique Séquentielle

Verrou S-R avec “enable”

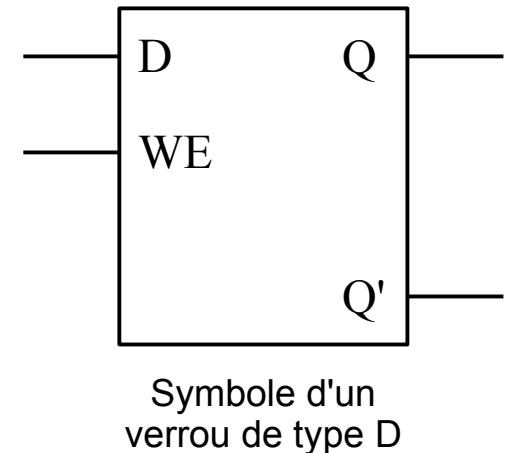
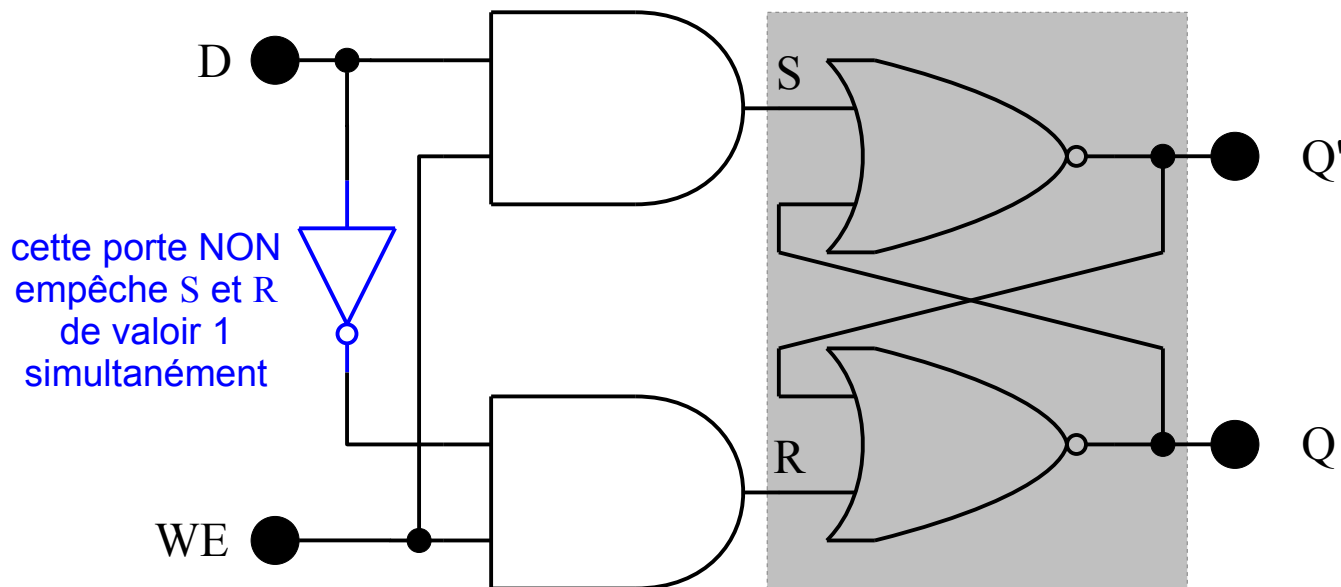
- Un verrou S-R réagit dès que les entrées S ou R sont activées. Il est parfois nécessaire que cette prise en compte ne soit possible que lorsqu'un signal “enable” le permet explicitement.
- Cela peut être réalisé par l'ajout de deux portes ET.



Logique Séquentielle

Verrou de type D

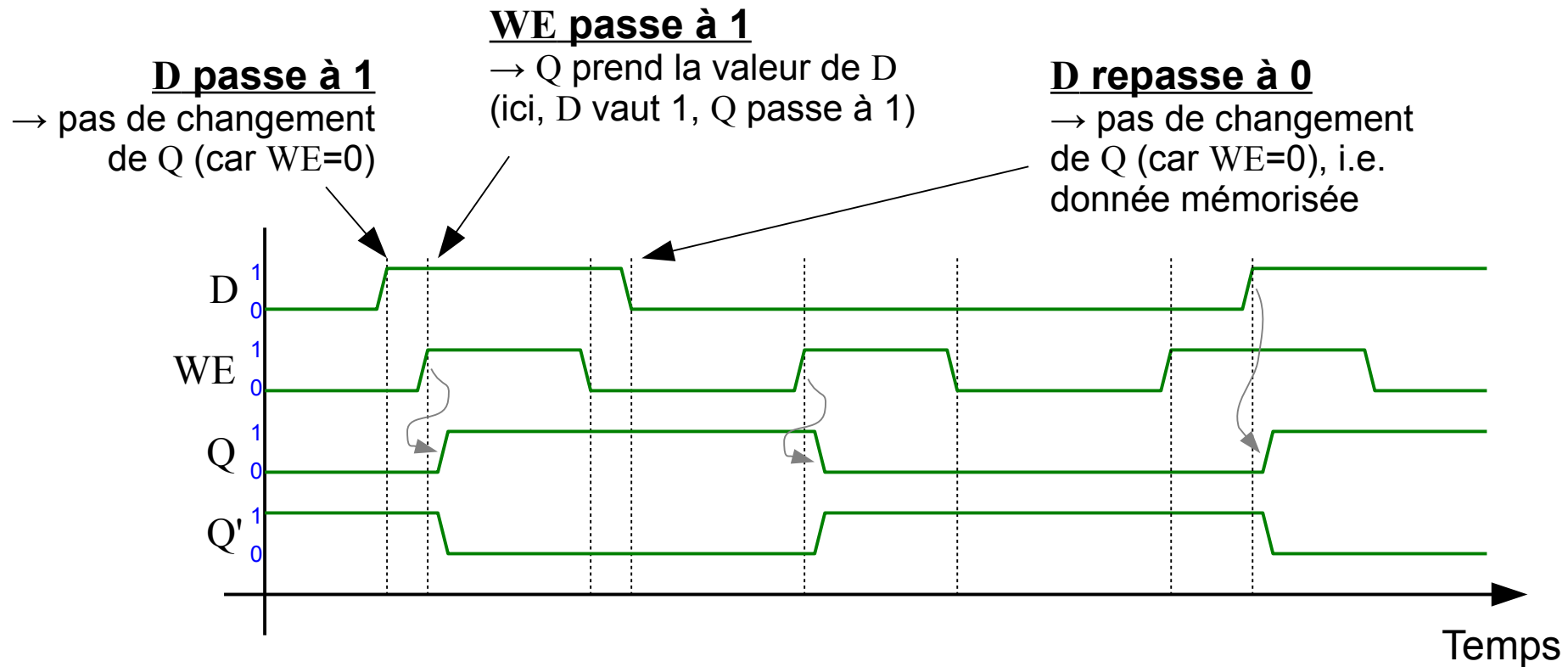
- Le **verrou de type D** permet d'éviter le problème de méta-stabilité du verrou S-R. De plus, il correspond mieux à l'idée d'une cellule permettant de mémoriser 1 bit car il possède une entrée D qui représente la valeur de la donnée à stocker.
- La donnée présentée est prise en compte lorsqu'une seconde entrée WE (*write enable*) est activée.



Logique Séquentielle

Verrou de type D

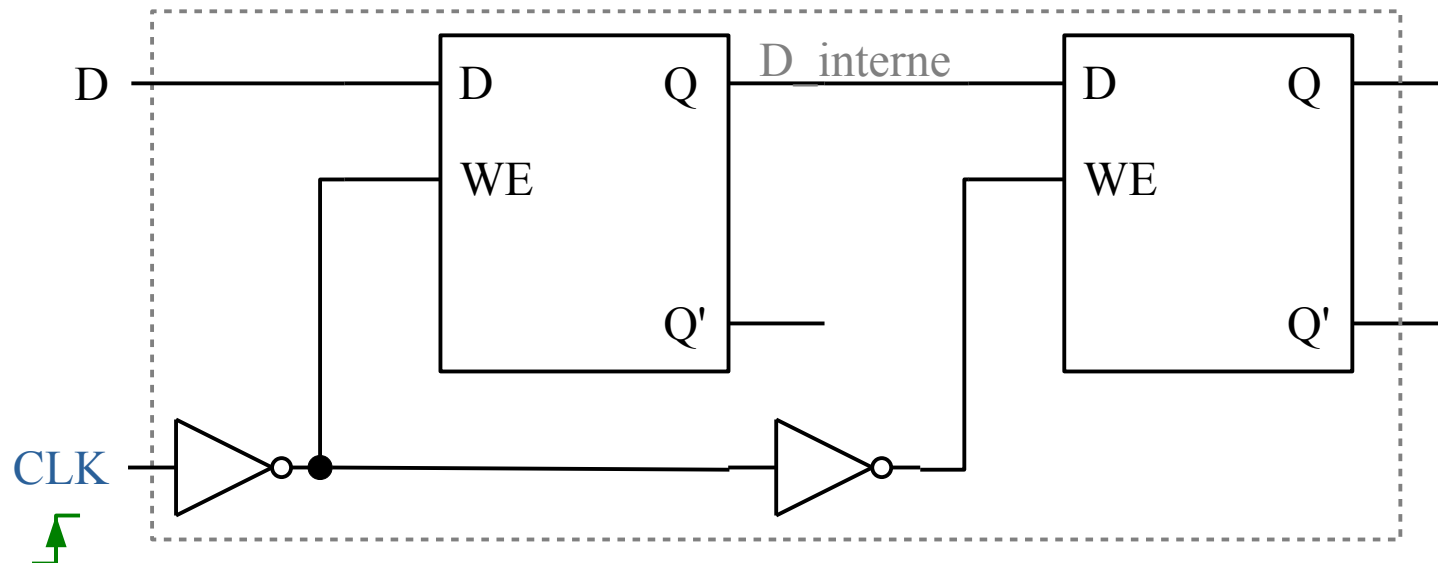
- Le diagramme suivant illustre le fonctionnement d'un verrou de type D.



Logique Séquentielle

Bascule bistable de type D

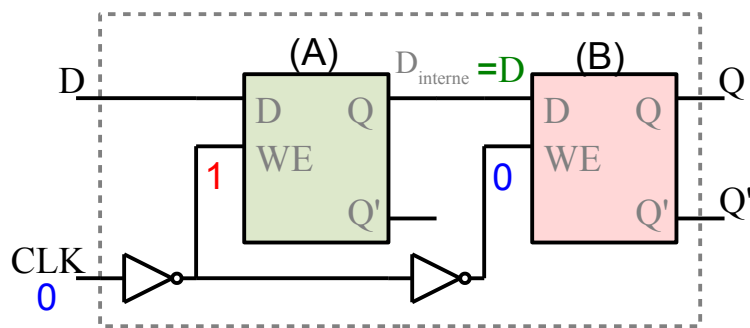
- La **bascule bistable de type D** (*D flip-flop*) a pour objectif le stockage d'un bit. A la différence du verrou de type D, la bascule bistable change son contenu uniquement lorsque le signal **CLK** change de bas vers haut. On dit aussi que la bascule bistable de type D **échantillonne sur le flanc montant** de **CLK** (*edge-triggered*).
- La bascule bistable de type D contient deux verrous de type D.



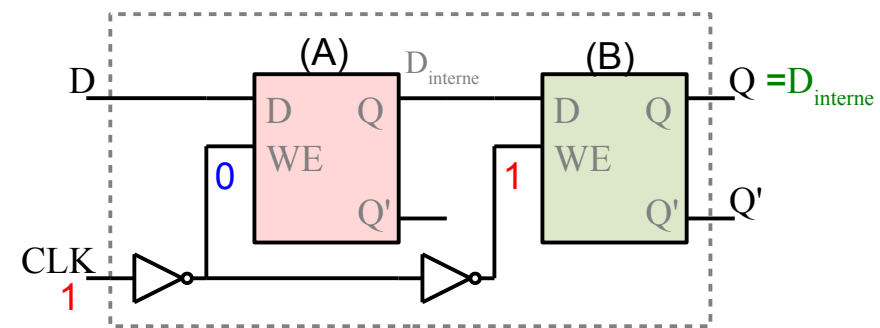
Logique Séquentielle

Bascule bistable de type D

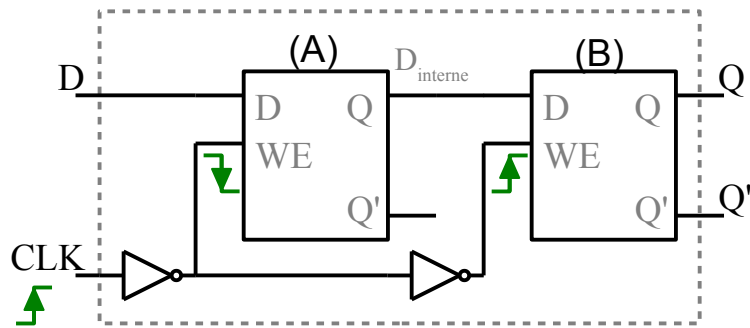
- Distinguons plusieurs états et transitions de CLK



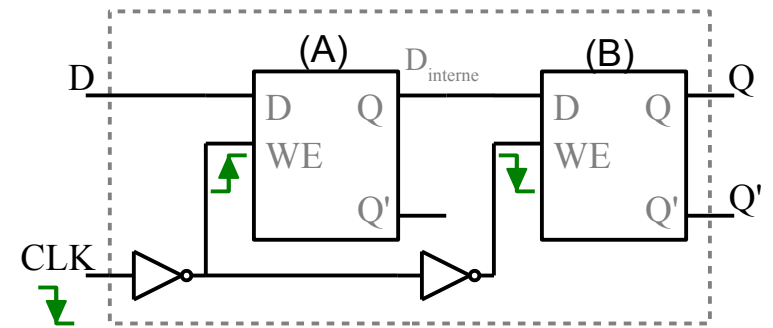
Verrou A : suit l'état dicté par D
Verrou B : conserve son état précédent (WE=0)



Verrou A : conserve son état précédent (WE=0)
Verrou B : suit l'état D_{interne} (ne change pas)



Verrou A : bloque son état sur $Q = D_{\text{interne}} = D$
au moment de la transition (échantillonne)

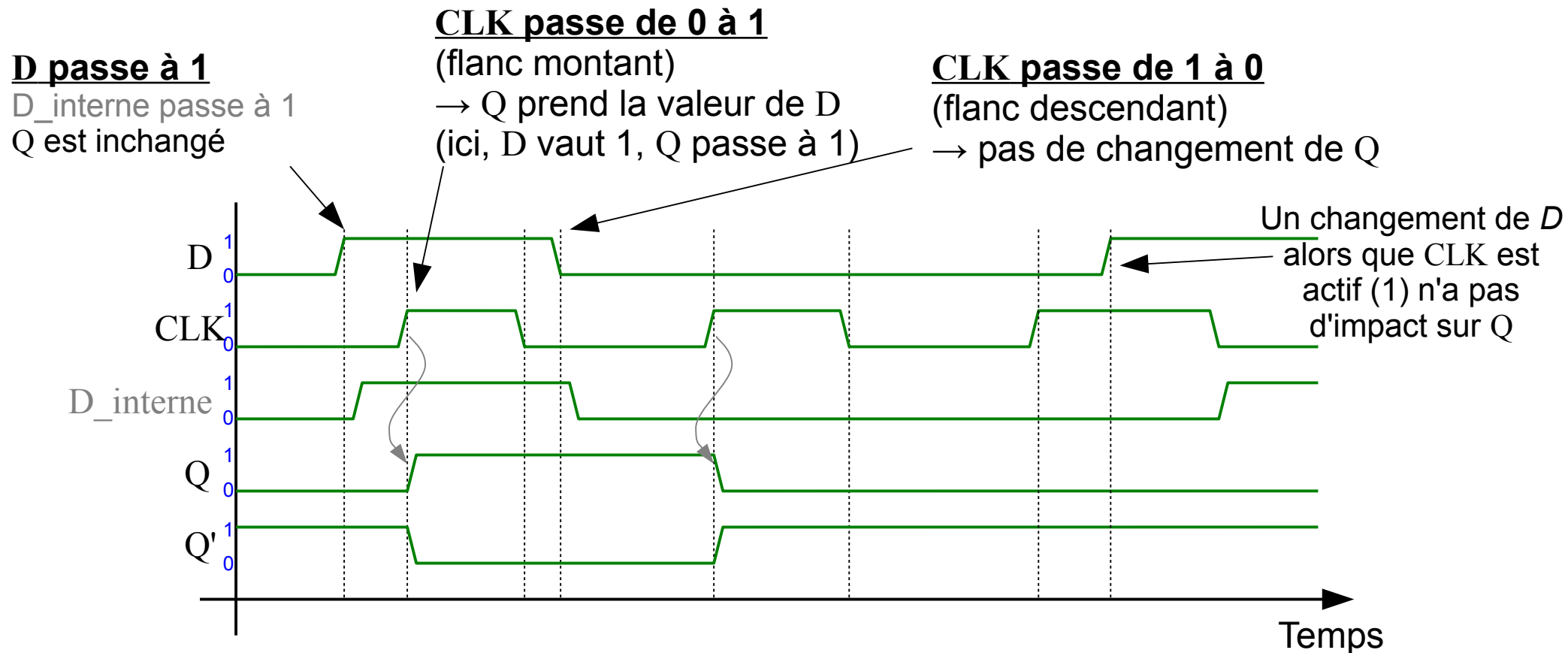


Verrou B : bloque son état sur $Q = D_{\text{interne}}$
au moment de la transition (échantillonne)

Logique Séquentielle

Bascule bistable de type D

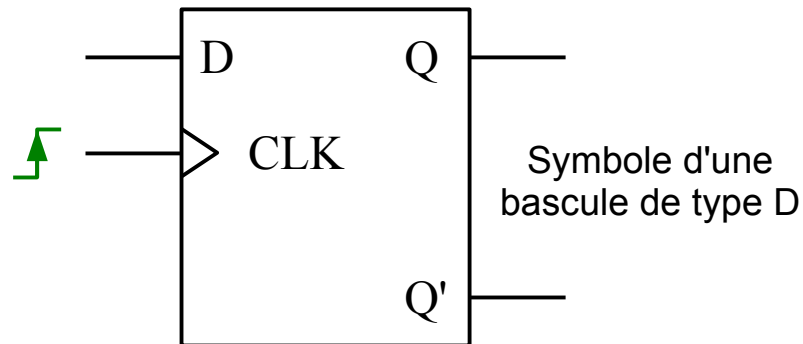
- Le diagramme suivant illustre le fonctionnement d'une bascule bistable de type D.



Logique Séquentielle

Bascule bistable de type D

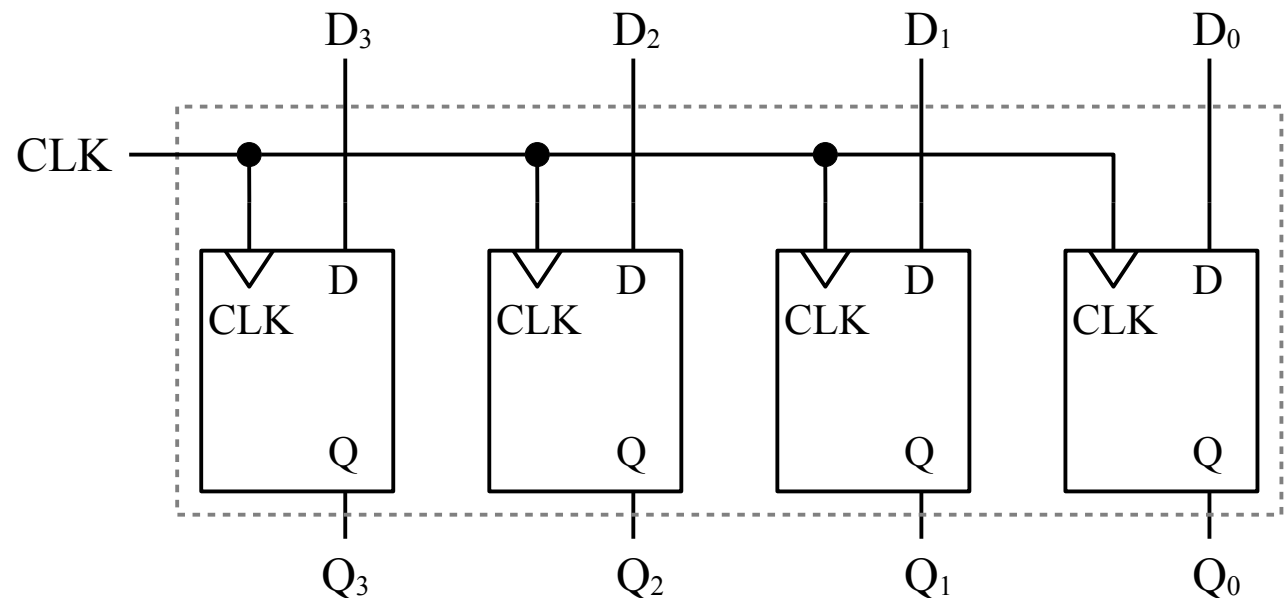
- Les bascules bistables sont prévues pour un fonctionnement **synchronisé avec une horloge**, contrairement aux verrous qui prennent en compte leurs entrées à tout moment.
- La bascule bistable de type D que nous venons d'étudier prend en compte la donnée D sur un **flanc montant** de **CLK** (*positive edge-triggered*) mais il existe des versions fonctionnant sur un **flanc descendant** de **CLK** (*negative edge-triggered*).
- Le symbole utilisé pour représenter une bascule bistable de type D est le suivant. Le triangle sur l'entrée **CLK** indique la prise en compte sur un flanc du signal.



Logique Séquentielle

Registre

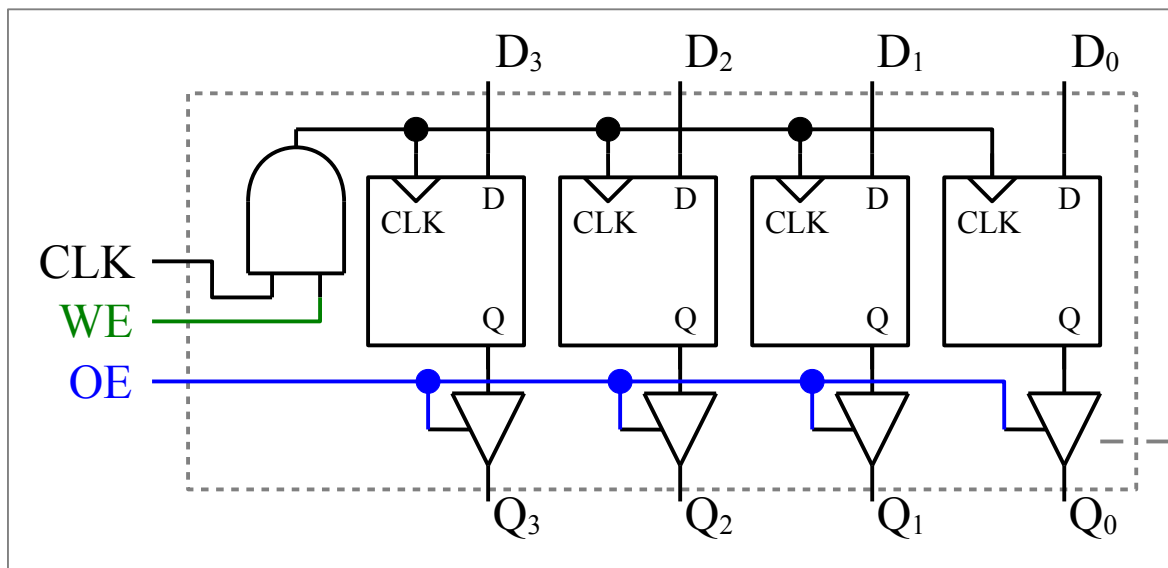
- Un **registre** est un élément permettant de stocker N bits. Il repose sur l'utilisation de N bascules bistables (par exemple de type D). Les N bits du registre constituent un **mot**. L'accès au registre est effectué par mot, i.e. tous les bits du registre sont écrits/ lus simultanément (les entrées CLK de toutes les bascules sont reliées entre elles).
- Les registres ont généralement des tailles de 8, 16, 32 et 64 bits.
- Par exemple:
registre de taille $N=4$ bits.



Logique Séquentielle

Registre

- Un registre contient généralement des signaux de contrôle supplémentaires :
 - **WE** (*write enable*) : l'écriture sur un flanc montant de l'horloge n'a lieu que si WE=1
 - **OE** (*output enable*) : le contenu des cellules du registre (bascules) n'est présenté en sortie que si OE=1. Dans le cas contraire, les sorties sont isolées des cellules.



Ce symbole figure un “tampon” (*buffer*) qui permet d'isoler la sortie Q_i de la cellule i en fonction d'un signal (OE). L'état isolé est souvent dénommé “haute-impédance” (*high-Z* or *tri-state*).

Logique Séquentielle

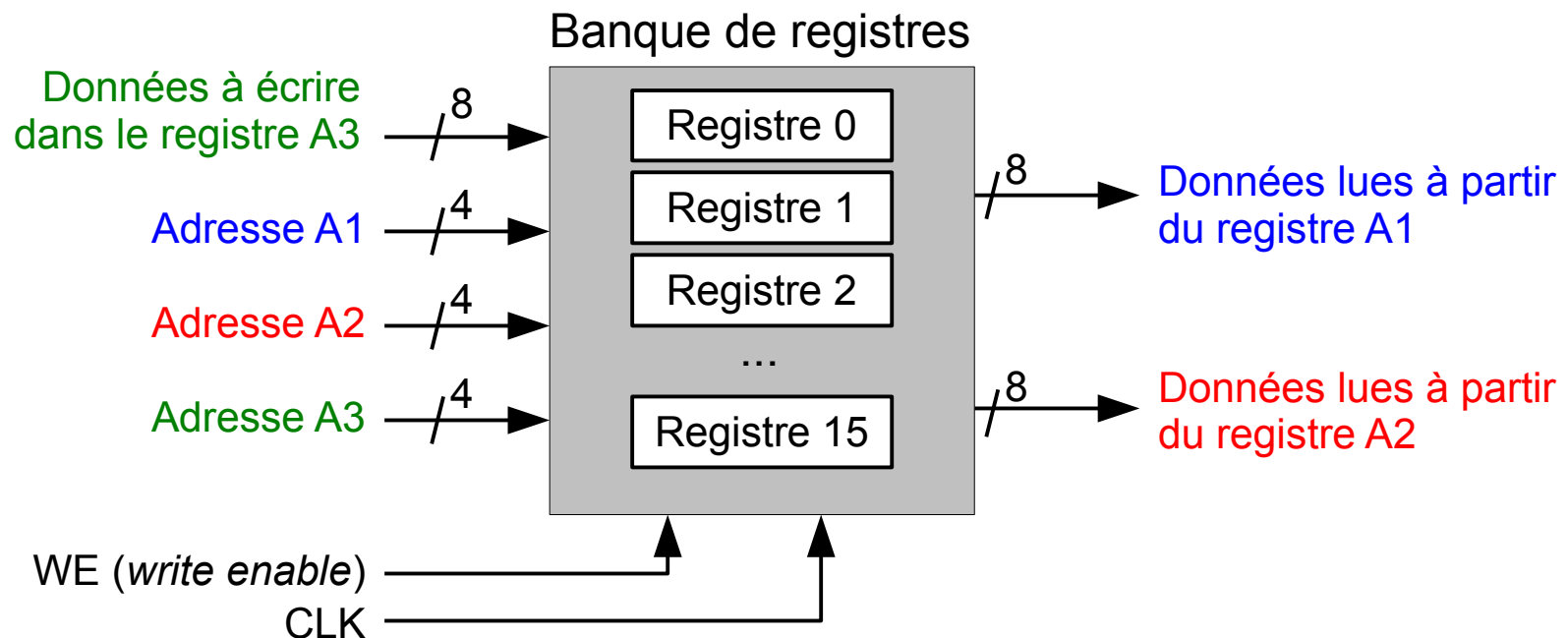
Banque de registres

- Un processeur contient un ensemble de plusieurs **registres d'usage général** (**GPR** – *General Purpose Registers* ou *scratchpad registers*). Les opérations réalisées par le processeur utilisent ces registres comme source et destination. L'ensemble des registres et les mécanismes qui permettent d'y accéder est appelé **banque de registres** (*register file*).
- Une banque de registres permet généralement la lecture et l'écriture de plusieurs registres simultanément. On parle de **mémoire multi-ports**.
- Chaque registre est identifié par un nombre appelé index ou **adresse**.

Logique Séquentielle

Banque de 16 registres de 8 bits

- Deux registres peuvent être lus simultanément.
- Un registre peut être écrit.
- Il y a 16 registres → une adresse est encodée sur 4 bits.
- Les opérations de lecture/écriture se font de façon synchrone : il n'y a qu'une entrée pour un signal d'horloge.



Logique Séquentielle

Banque de 16 registres de 8 bits

- Pour l'écriture, l'adresse (**A3**) permet d'activer l'entrée **WE** d'un seul registre, au travers d'un décodeur binaire 4:16.
- L'écriture a lieu lorsque **WE**=1 et lors du flanc montant de **CLK**.
- Pour la lecture, les adresses (**A1** et **A2**) contrôlent chacune un multiplexeur 16:1.

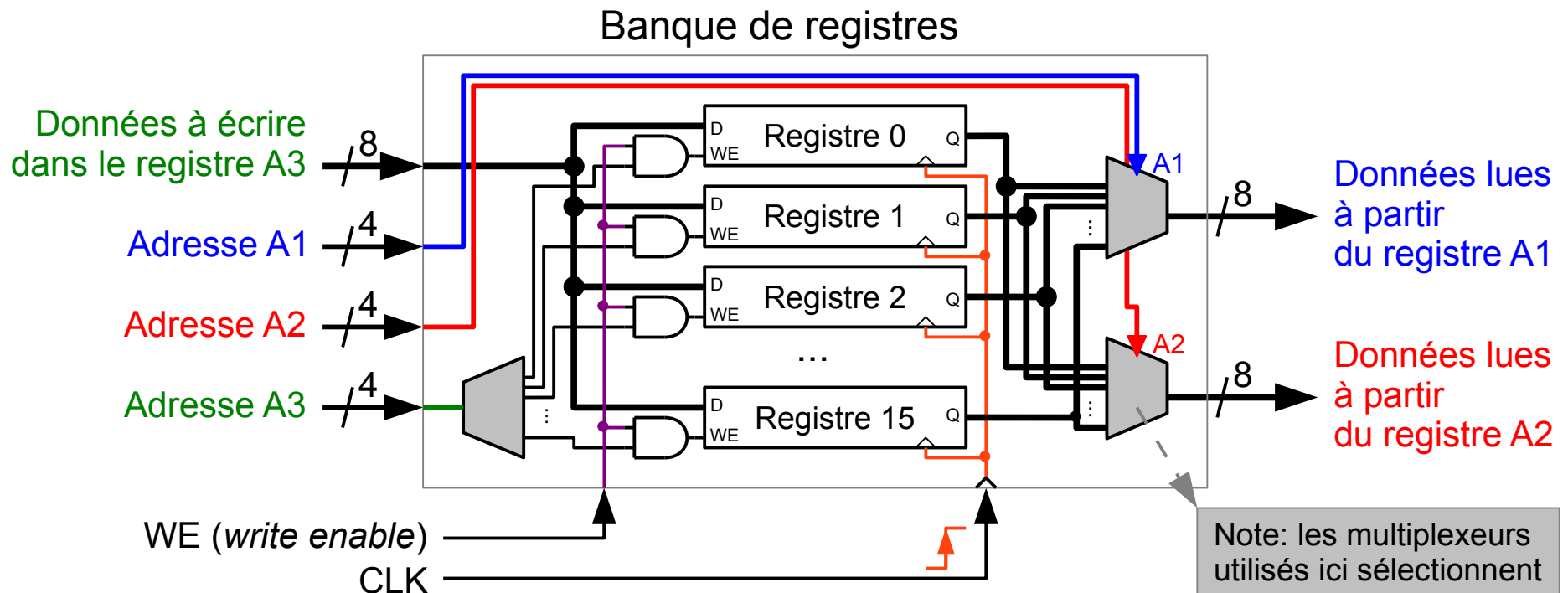


Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

- Principes
- Verrou, Bascule bistable, Registre

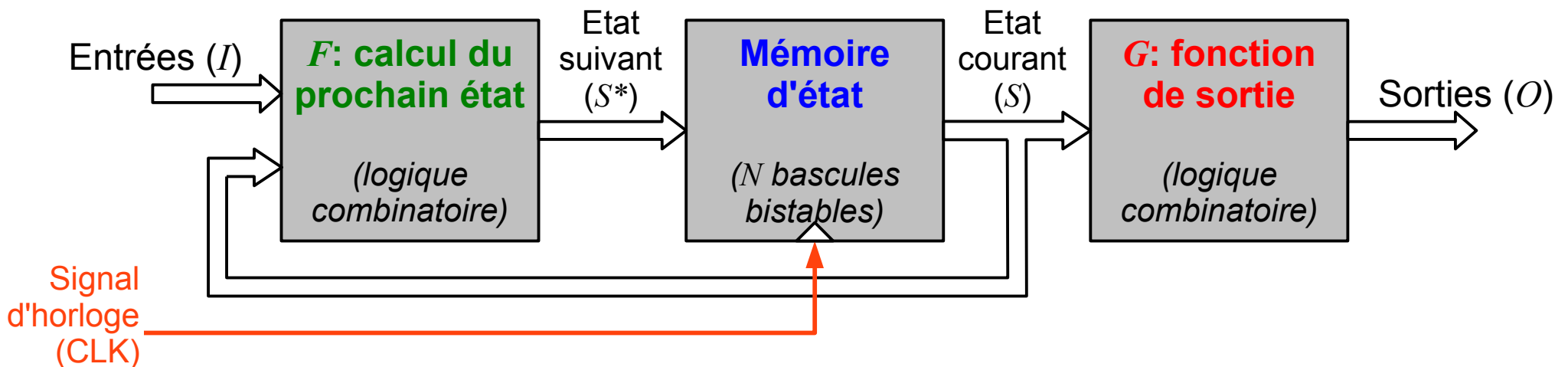
➡ Machines à états

- Mémoire

Machines à états

Principes

- Une **machine à états** est un circuit logique séquentiel. Ses sorties dépendent de ses entrées et d'un état mémorisé.
- Nous étudions les machines **à états synchrones avec horloge**. Une telle machine à états conserve son état courant dans une **mémoire d'état** composée de N bascules bistables (2^N états sont possibles).
- Le changement d'état est réalisé **de façon synchrone**, lors de la réception d'un **signal d'horloge**.

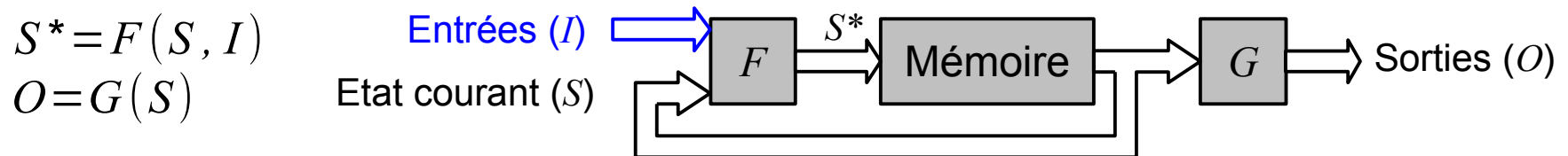


Machines à états

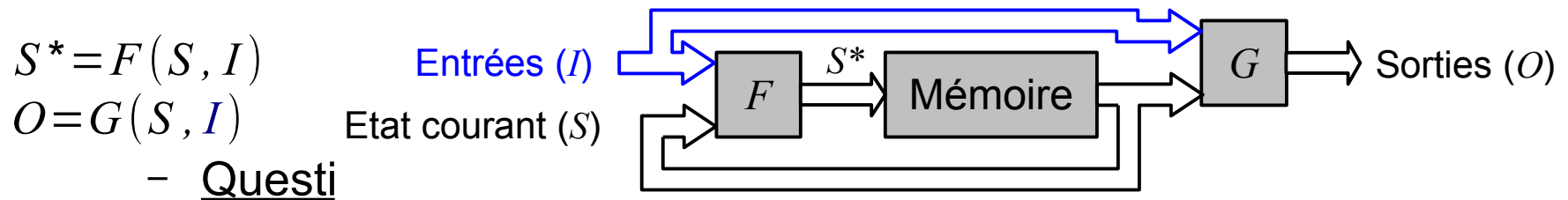
Définition

- **Machines à états**

- On distingue les machines à états de Moore et les machines à états de Mealy.
- **Moore** : les sorties ne dépendent que de l'état courant.



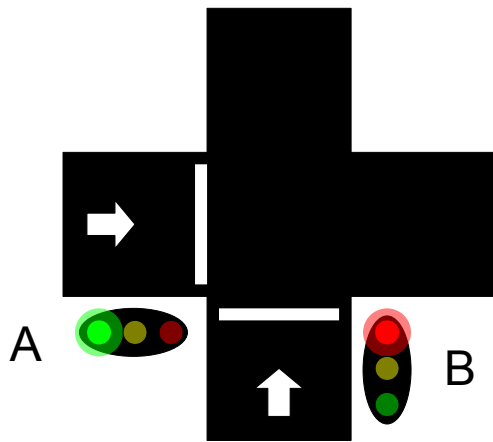
- **Mealy** : les sorties dépendent de l'état courant **et des entrées**.



Machines à états

Contrôleur de feux de signalisation

- Une application simple et classique des machine à états est l'implémentation d'un contrôleur pour les feux de signalisation utilisés au croisement de deux routes.

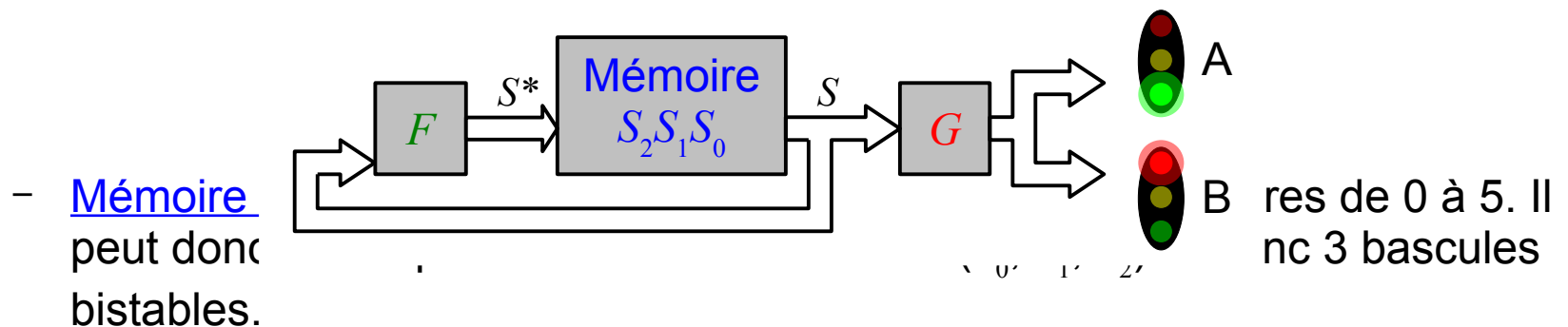


Etat	Feu A			Feu B		
	Rouge	Orange	Vert	Rouge	Orange	Vert
0	1	0	0	1	0	0
1	0	0	1	1	0	0
2	0	1	0	1	0	0
3	1	0	0	1	0	0
4	1	0	0	0	0	1
5	1	0	0	0	1	0

Machines à états

Contrôleur de feux de signalisation

- Le contrôleur de feux de signalisation peut être implémenté avec une machine à états de Moore, sans entrée.



- Fonction F : l'état suivant (S^*) ne dépend que de l'état courant (S).
- Fonction G : la sortie est obtenue sur base de l'état courant uniquement (Moore).

Machines à états

Contrôleur de feux de signalisation

- Etablissons la table des états et transitions (vers les états suivants).

Etat			Etat suivant			Feu A			Feu B		
S_0	S_1	S_2	S_0^*	S_1^*	S_2^*	Rouge	Orange	Vert	Rouge	Orange	Vert
0	0	0	1	0	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	0
1	1	0	0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0	1
1	0	1	0	0	0	1	0	0	0	1	0

En utilisant une somme canonique, il est aisé de dériver la fonction F calculant l'état suivant.

$$\begin{aligned}
 S_0^* &= S_0' \cdot S_1' \cdot S_2' + S_0' \cdot S_1 \cdot S_2' + S_0' \cdot S_1' \cdot S_2 = \sum_{S_0, S_1, S_2} (0, 2, 4) \\
 S_1^* &= S_0 \cdot S_1' \cdot S_2' + S_0' \cdot S_1 \cdot S_2' = \sum_{S_0, S_1, S_2} (1, 2) \\
 S_2^* &= S_0 \cdot S_1 \cdot S_2' + S_0' \cdot S_1' \cdot S_2 = \sum_{S_0, S_1, S_2} (3, 4)
 \end{aligned}$$

Machines à états

Contrôleur de feux de signalisation

- Etablissons la table des états et transitions.

Etat			Etat suivant			Feu A			Feu B		
S_0	S_1	S_2	S_0^*	S_1^*	S_2^*	Rouge	Orange	Vert	Rouge	Orange	Vert
0	0	0	1	0	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	0
1	1	0	0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0	1
1	0	1	0	0	0	1	0	0	0	1	0

De la même façon,
il est possible
d'exprimer la fonction **R**
donnant les sorties
en fonction de l'état courant.

$$A_R = \sum_{S_0, S_1, S_2} (0, 3, 4, 5)$$

$$A_O = \sum_{S_0, S_1, S_2} (2)$$

$$A_V = \sum_{S_0, S_1, S_2} (1)$$

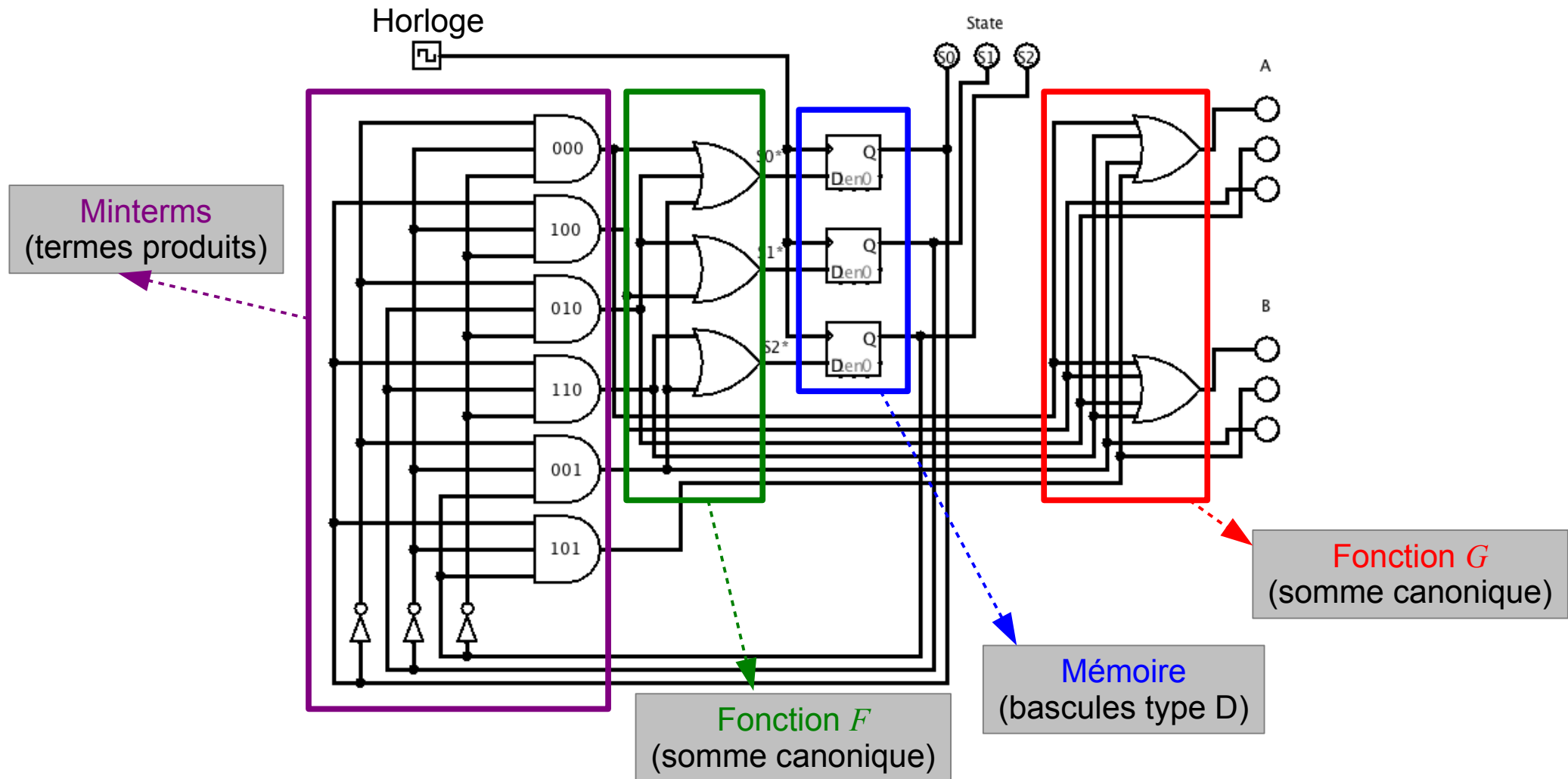
$$B_R = \sum_{S_0, S_1, S_2} (0, 1, 2, 3)$$

$$B_O = \sum_{S_0, S_1, S_2} (5)$$

$$B_V = \sum_{S_0, S_1, S_2} (4)$$

Machines à états

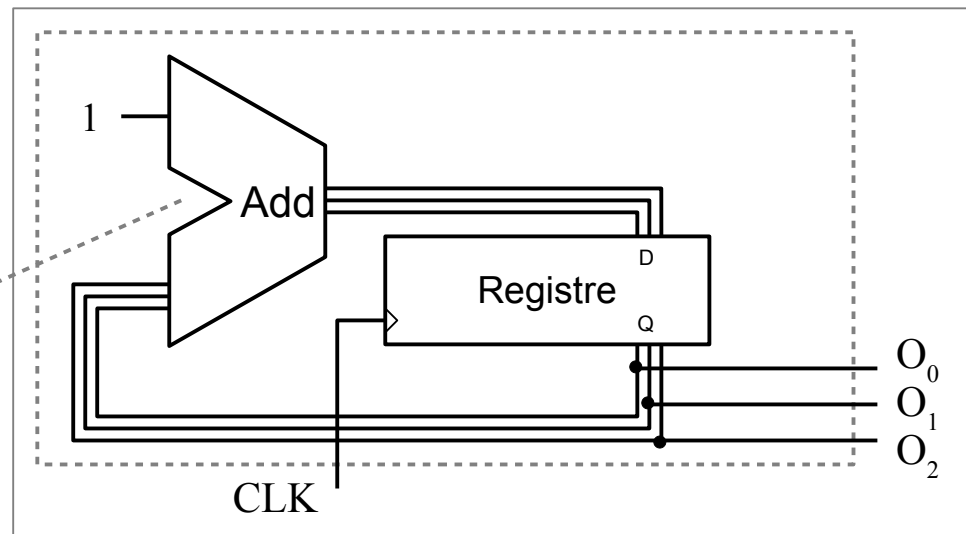
Contrôleur de feux de signalisation



Machines à états

Compteur

- Un **compteur** est une machine à états extrêmement utile. Un compteur a des applications dans le fonctionnement d'un processeur, de timers, etc.
- Un compteur est composé d'un état sur N bits qui représente une valeur comprise entre 0 et $2^N - 1$. La logique calculant l'état suivant est celle de l'addition modulo 2^N .
- Exemple :
compteur 3 bits



Note: même si un additionneur complet est schématisé, l'addition avec une constante (ici 1) peut être réalisée de façon plus efficace autrement (comment?).

Table des Matières

Introduction

- Objectifs
- Circuits logiques
- Algèbre de Commutation
- Portes logiques

Logique combinatoire

- Principes
- Décodeur, Multiplexeur, Additionneur, ALU

Logique séquentielle

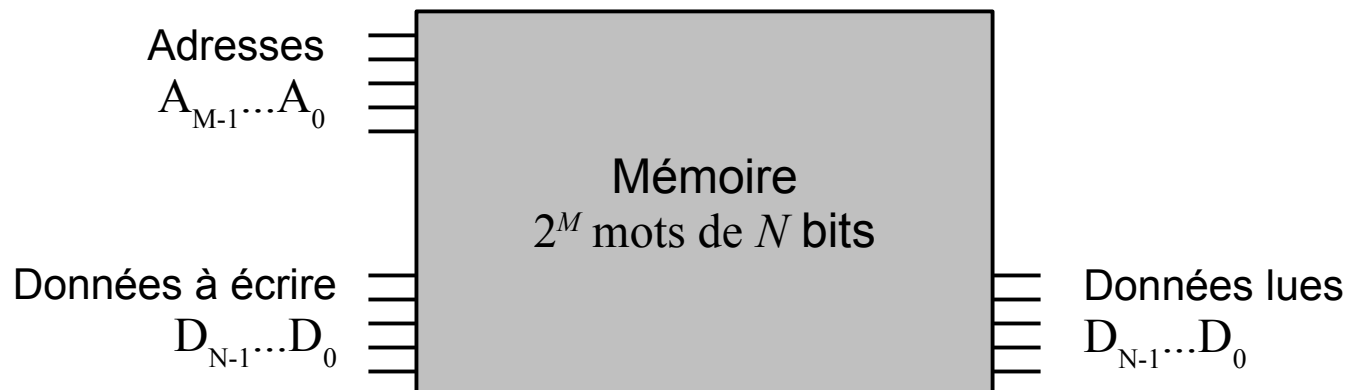
- Principes
- Verrou, Bascule bistable, Registre
- Machines à états

➡ Mémoire

Mémoire

Mémoire

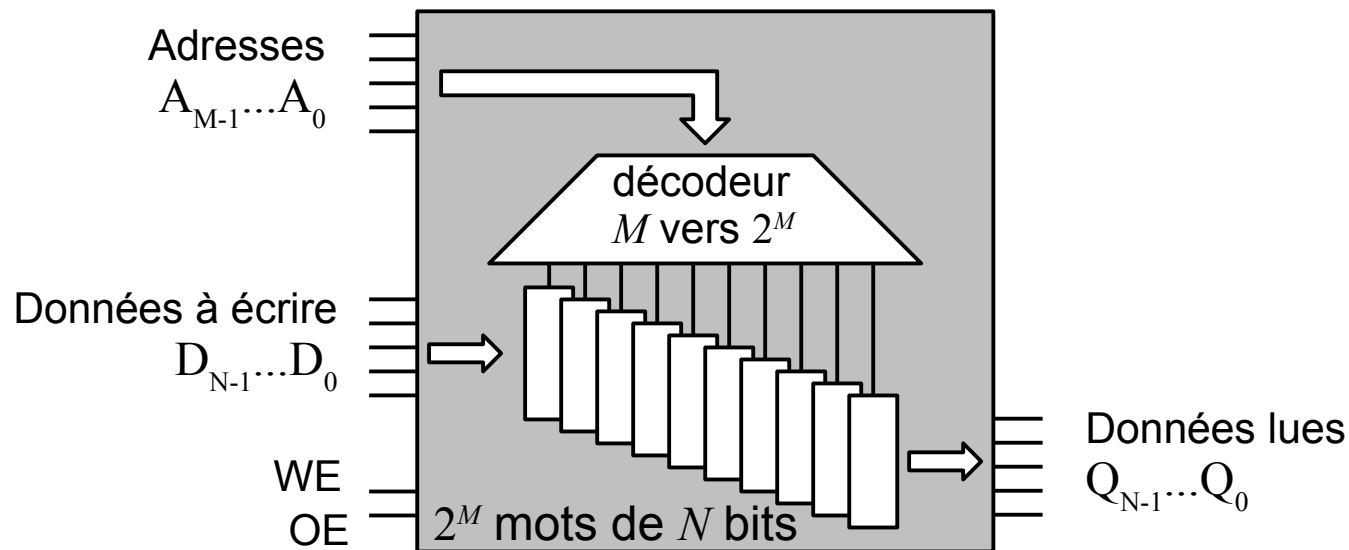
- Une **mémoire à accès aléatoire** (*random access memory - RAM*) permet de stocker 2^M mots de N bits, ce qui représente un total de $2^M \cdot N$ bits. Chaque mot est accessible individuellement, dans un ordre quelconque (accès aléatoire – *random access*).
- Une **adresse** désigne quel mot doit être lu ou écrit. Une adresse est fournie à la mémoire sous forme d'un mot binaire. Une adresse de M bits permet d'identifier un mot parmi 2^M .
- Chaque accès lit ou écrit un mot de N bits à la fois. La taille d'un mot est la **largeur de la mémoire**.



Mémoire

Organisation

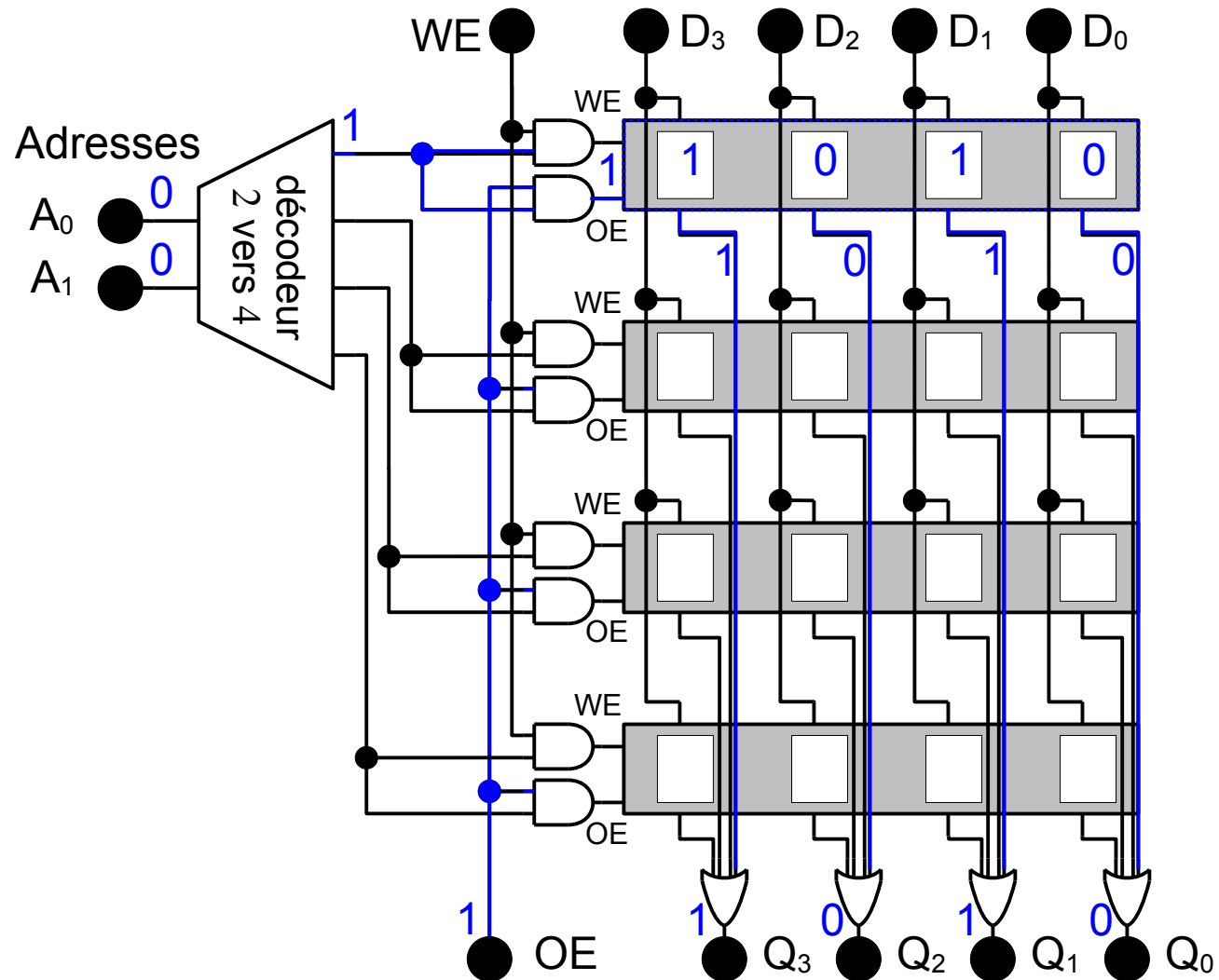
- Une RAM est souvent organisée en deux parties.
 - D'une part, une série de **cellules** (par exemple implémentées en utilisant des bascules bistables) organisées en 2^M mots de N bits.
 - D'autre part un **décodeur d'adresse** qui active un mot unique parmi 2^M .



Mémoire

Implémentation

- Exemple : mémoire de 4 x 4 bits



Mémoire

Types de Mémoires

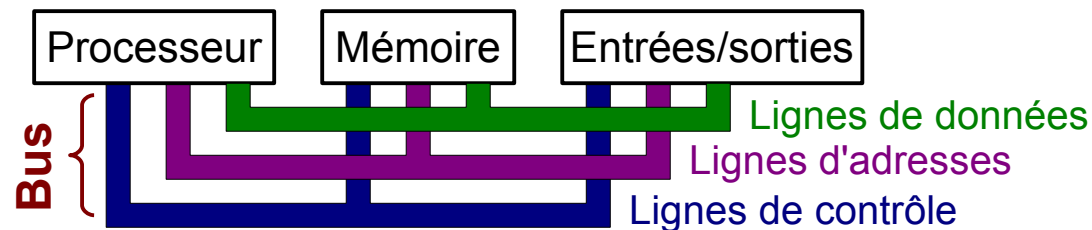
- Il existe une grande variété de mémoires.
- **RAM** – Mémoire à accès aléatoire (*random access memory*)
 - **SRAM** : RAM statique, cellule implémentée uniquement avec des semiconducteurs
 - **DRAM** : RAM dynamique, cellule implémentée avec notamment un condensateur
 - **SDRAM** : RAM dynamique synchrone (p.ex. DDR_x)
 - ...
- **ROM** - Mémoire en lecture seule (*read only memory*)
 - Typiquement utilisées pour contenir le *firmware* d'un système (p.ex. BIOS sur les PC)
 - Certaines ROM sont programmables ou reprogrammables⁽¹⁾ (OT)PROM, EPROM, EEPROM, ...
 - ...

⁽¹⁾ OTPROM = *One-Time Programmable ROM*, EPROM = *Erasable Programmable ROM*, EEPROM = *Electrically Erasable Programmable ROM*

Mémoire

Bus

- Un **bus** est un système de communication permettant le transfert de données entre différentes entités d'un système informatique (p.ex. entre processeur, mémoire et entrées/sorties).

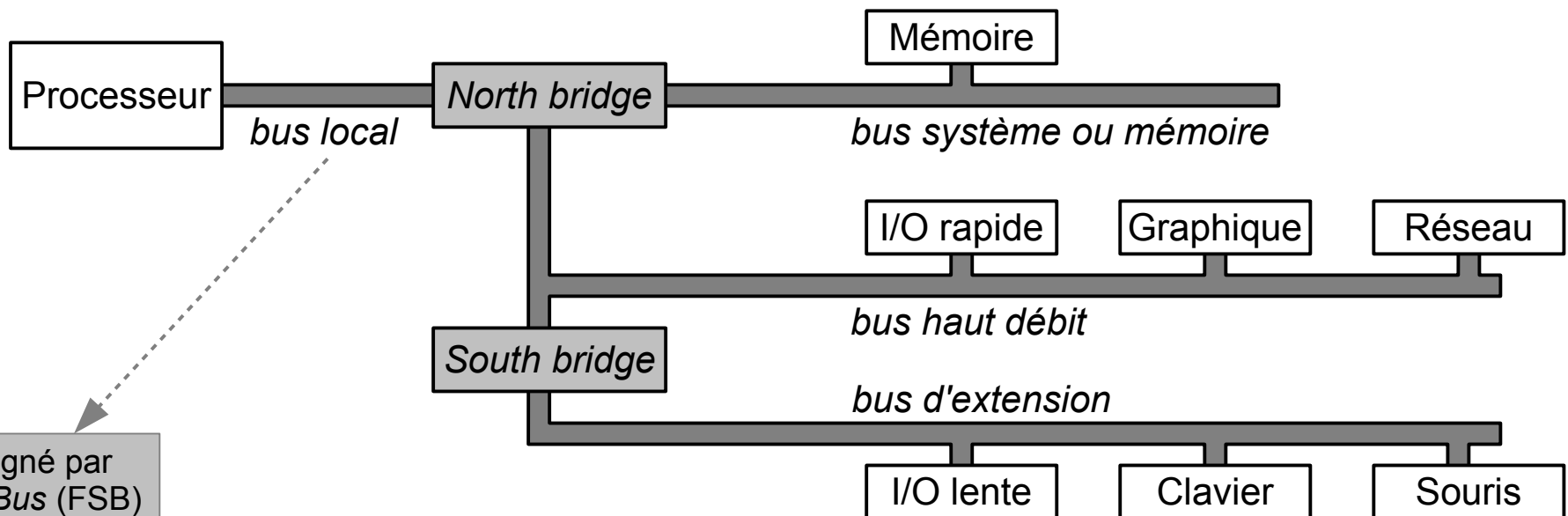


- Un bus regroupe de nombreux **signaux logiques** : adresses, données, signaux de contrôle (RW, WE, OE)
 - L'accès au bus est régi par un **protocole d'arbitrage**. Celui-ci définit quel composant peut accéder au bus à quel moment, afin d'éviter que plusieurs communications n'entre en collision.
 - Seuls les éléments respectant le protocole de ce bus peuvent être connectés ensemble.
- Exemples de bus
 - ISA, PCI, PCI Express, AGP, I²C, USB, QuickPathInterconnect (QPI), HyperTransport, ...

Mémoire

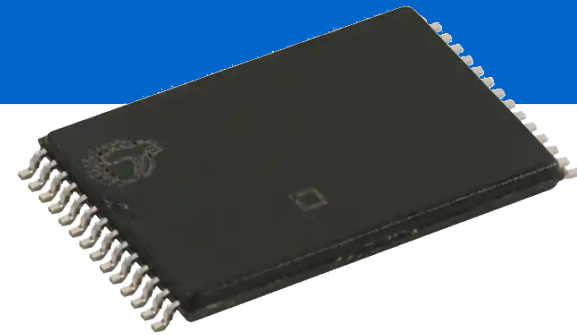
Hiérarchie de bus

- Des bus différents sont utilisés pour accéder à différents types de composants, en fonction de leur débit (nombre de bits par seconde) et de leur latence (temps de réaction en secondes).
- Ces bus sont souvent organisés de façon hiérarchique et interconnectés avec des **ponts** (*bridges*).

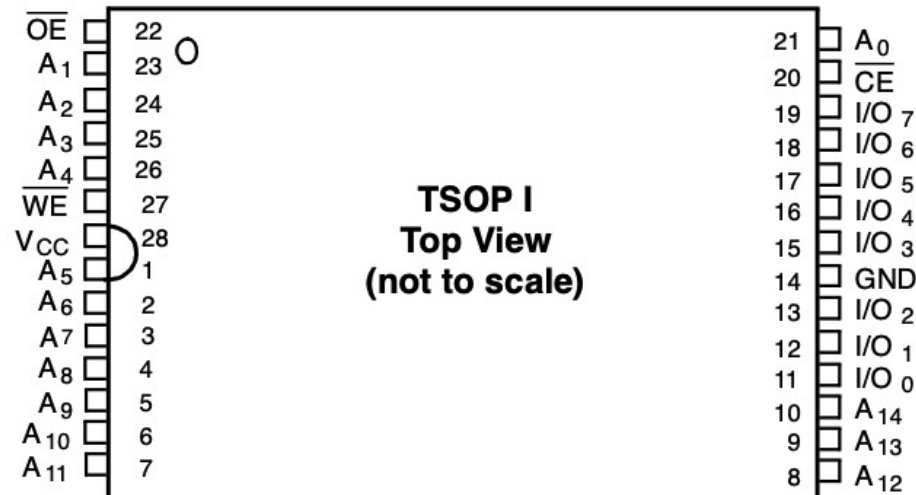
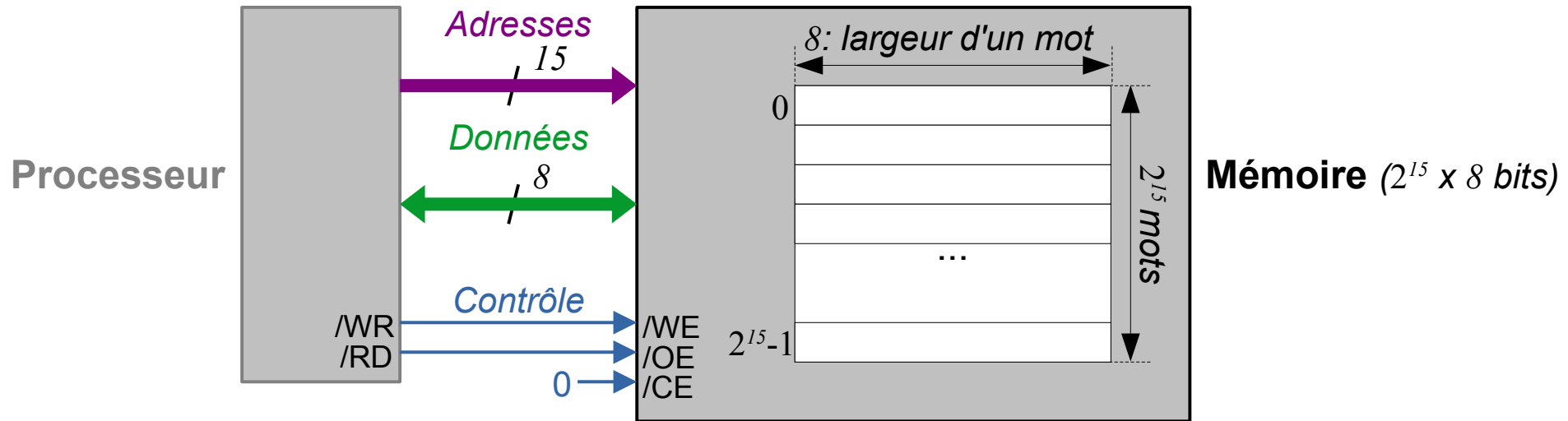


Mémoire

Etude de cas SRAM Cypress CY7C199

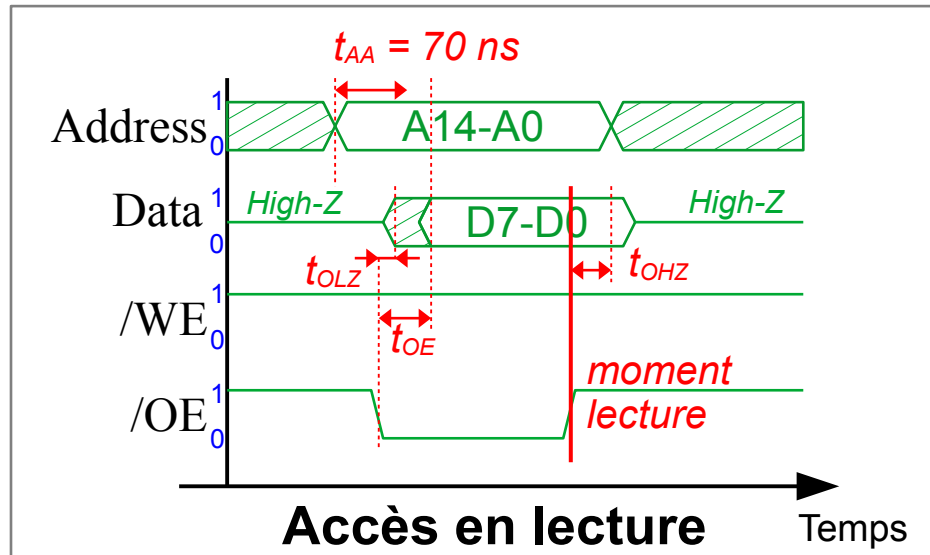


- Topologie : 32K x 8 (2^{15} mots de 8 bits)



Mémoire

Protocole d'accès à la mémoire

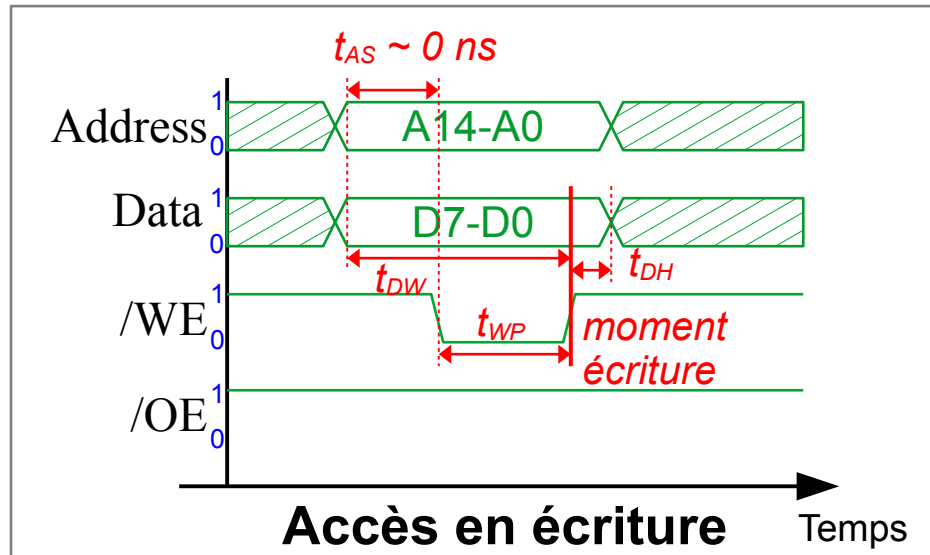


t_{AA} : address access
 t_{OLZ} : output low-Z
 t_{OE} :
 t_{OHZ} : output high-Z

- Lecture de la donnée D à l'adresse A
 1. Mémoire en lecture ($/WE=1$) : transfert mémoire \rightarrow processeur
 2. L'adresse est présentée à la mémoire sur le bus d'adresses.
 3. Activation sortie mémoire ($/OE=0$)
 4. Les données sont disponibles sur le bus de données.
 5. Désactivation sortie mémoire ($/OE=1$) – moment de la lecture

Mémoire

Protocole d'accès à la mémoire



t_{AS} : address setup
 t_{DW} : data-to-write
 t_{DH} : data hold
 t_{WP} : write pulse

- Ecriture de la donnée D à l'adresse A
 1. La sortie de la mémoire est désactivée ($/OE=1$)
 2. L'adresse A est présentée sur le bus d'adresses.
 3. La donnée D est présentée sur le bus de données.
 4. Le signal d'écriture est activé ($/WE=0$) durant t_{WP} .
 5. Le signal d'écriture est désactivé ($/WE=1$) – moment de l'écriture