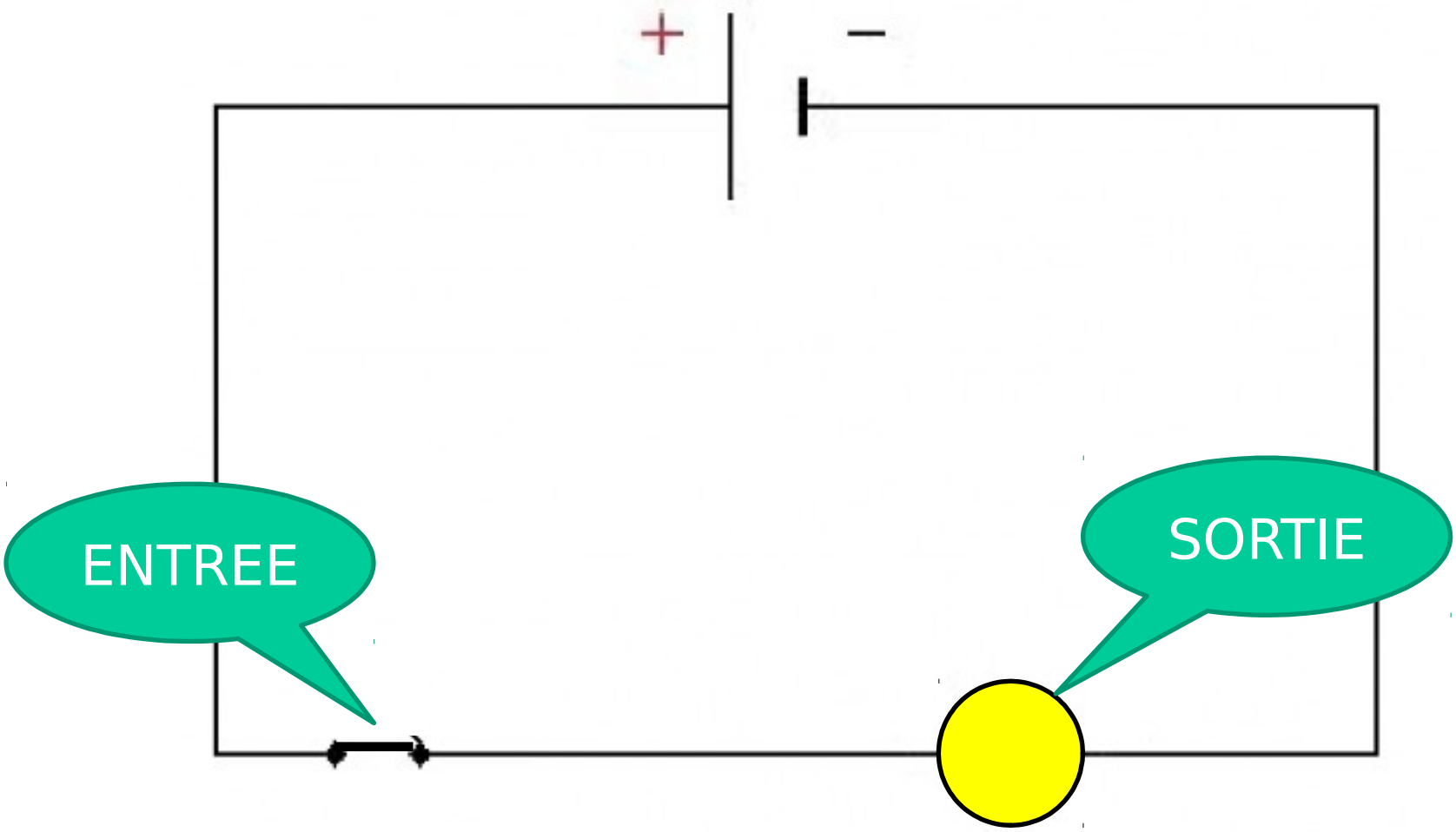
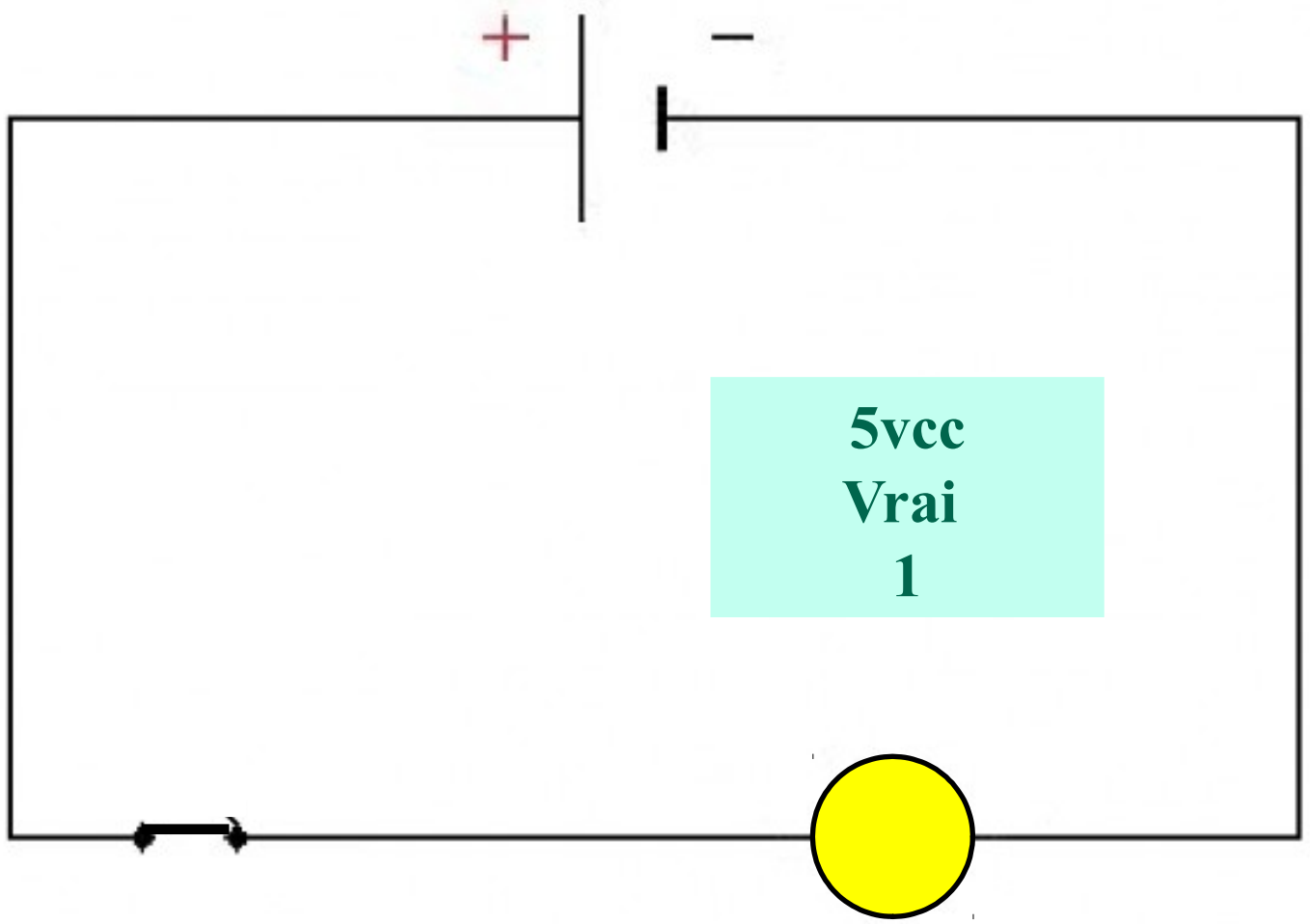
AE Cours 2

De l’électronique à l’informatique

Au commencement était l'électricité

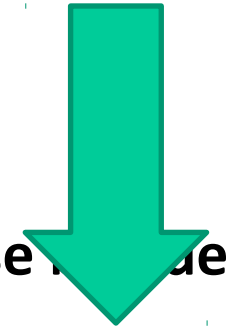


Au commencement était l'électricité



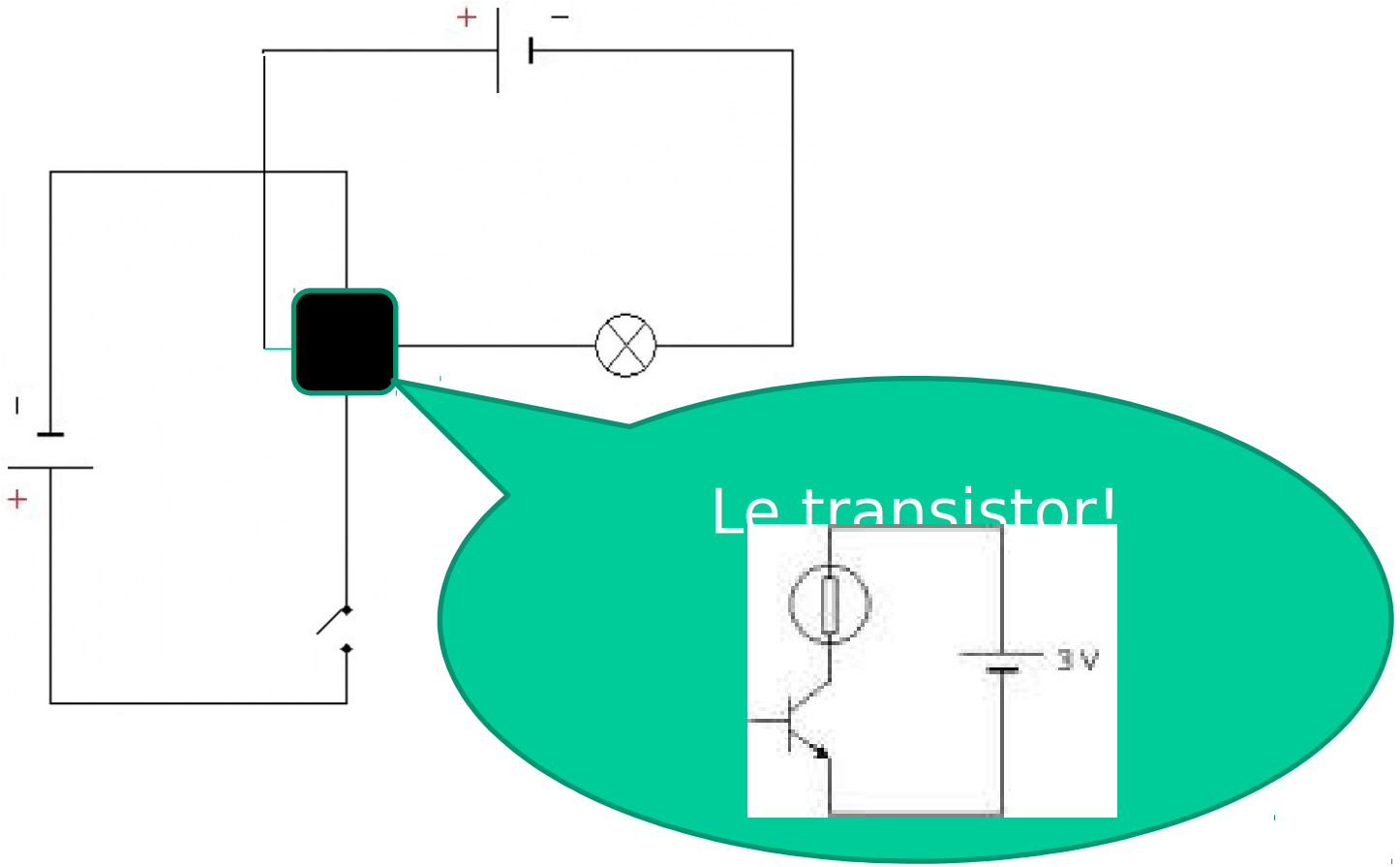
Fausses idées

- NON il n'y a pas des 0 et des 1 dans un ordinateur!



- Juste des électrons qui se [illegible]ent ou pas!!

Un interrupteur programmable!!



Mon premier « ordinateur »

- Un inverseur!
  - Si j envoie un courant rien ne sort
  - Si je n'envoie rien alors il sort un courant!

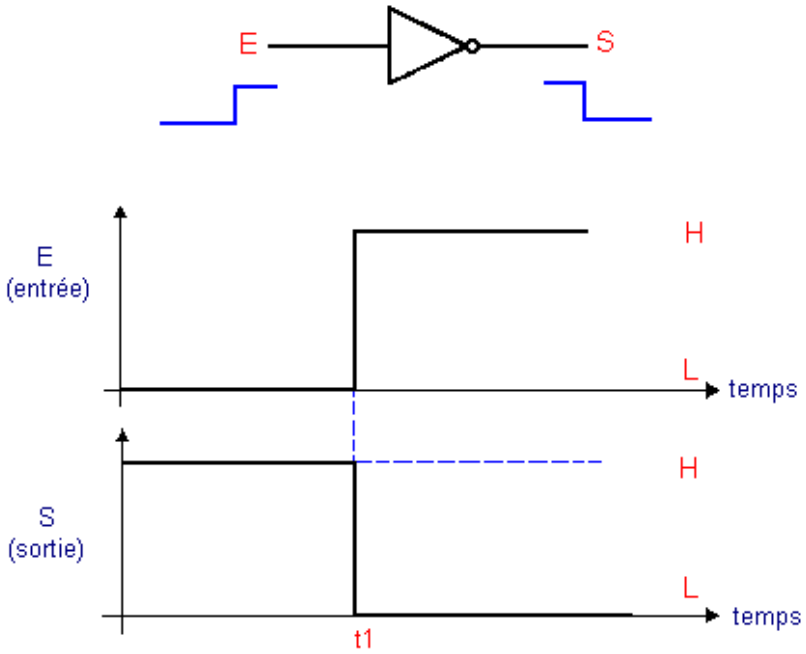
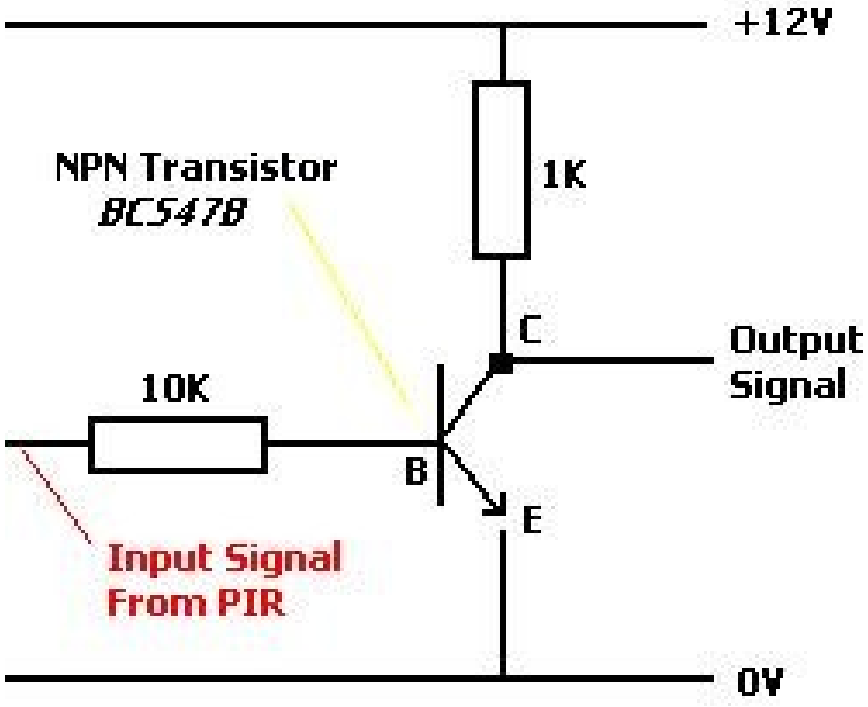
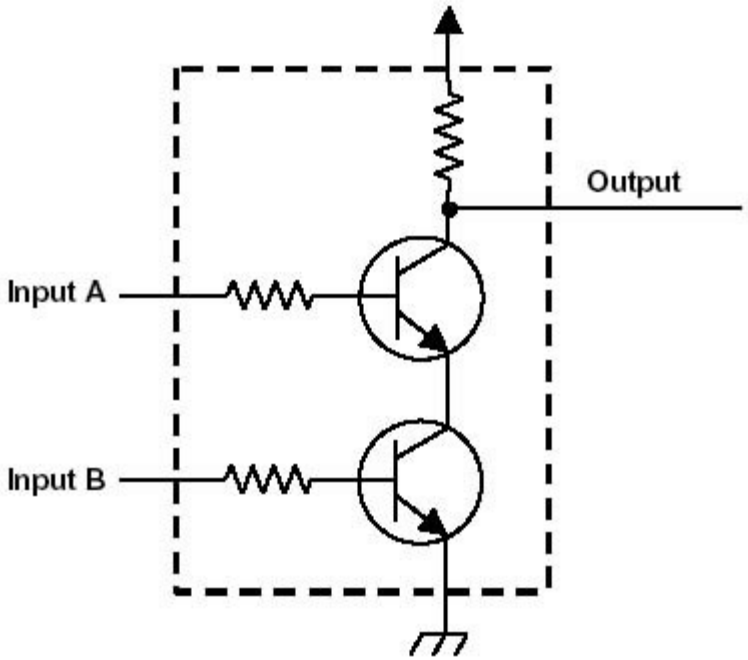


Fig. 28. - Signaux à l'entrée et à la sortie d'un inverseur.

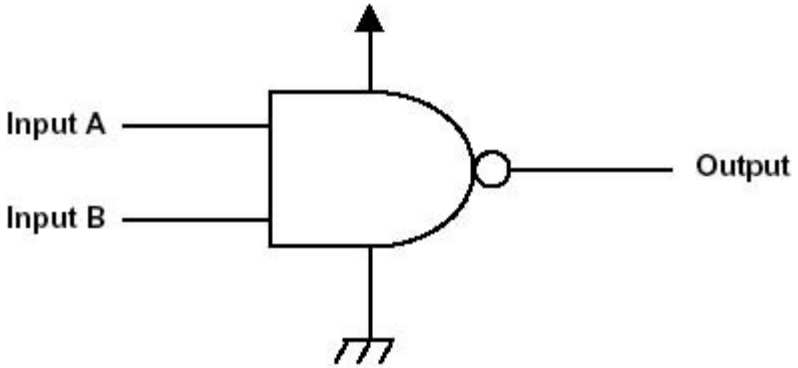


Avec deux entrées en séries



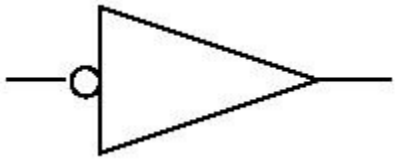
NAND:

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



Inverseur à base de Nand

NOT



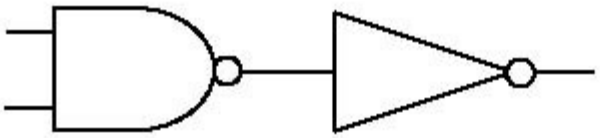
Input	Output
0	1
1	0

NOT

Input	Output
0	1
1	0

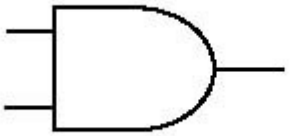
Un And

AND



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

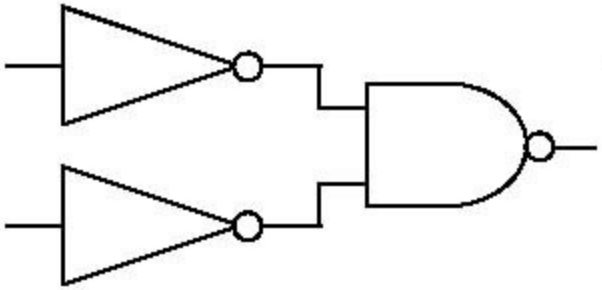
AND



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

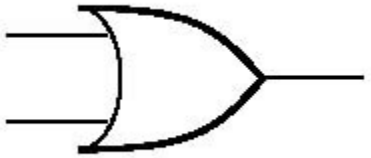
Le Or à partir du Nand

OR



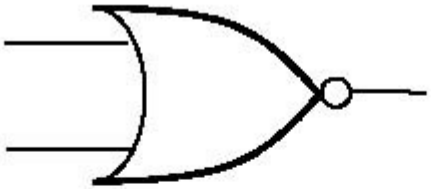
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

OR



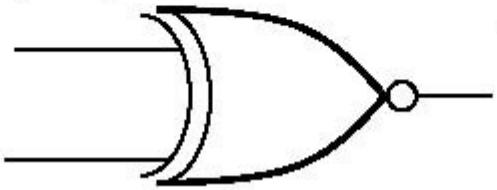
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOR



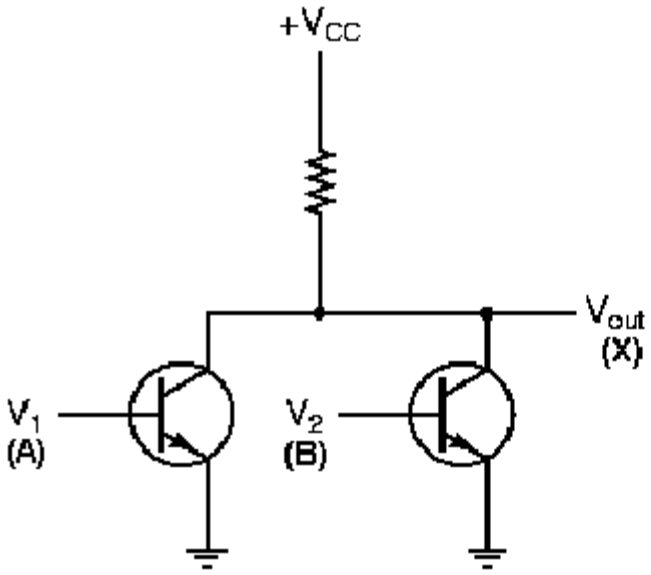
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

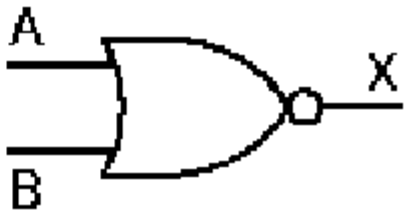
Exclusive NOR (XNOR)



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Le Nor en transistor

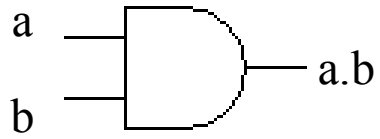


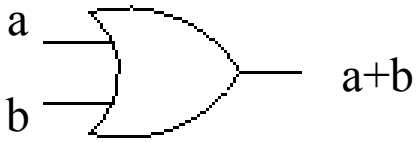


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Algèbre de Boole

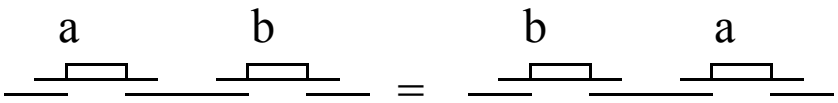
- Pour pouvoir manipuler des 0 et des 1, on a donc trois opérations :
  - Fonction **négation** (complémentation) **« NON »** (« NOT »)
    - noté avec une barre
    - 0 = 1 et 1 = 0
  - Fonction **conjonction « ET »** (« AND »)
    - noté « . »
    - 0.0 = 0.1 = 1.0 = 0 1.1 = 1
  - Fonction **disjonction « OU »** (« OR »)
    - noté « + »
    - 0+0 = 0 0+1 = 1+0 = 1+1 = 1



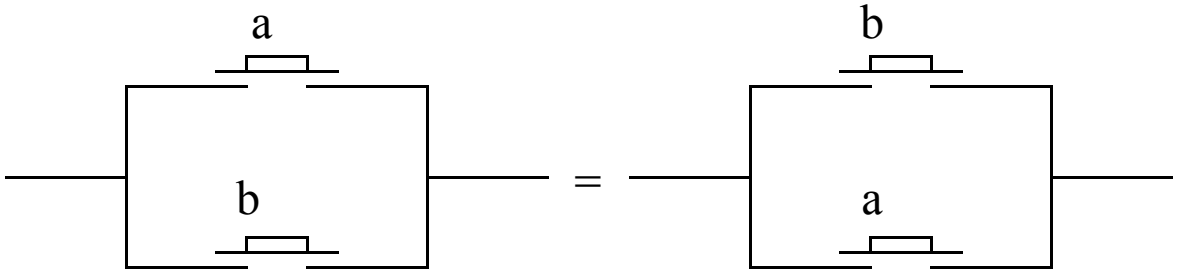


Axiomes de base (1/4)

- **Commutativité :**
  - **a.b = b.a**



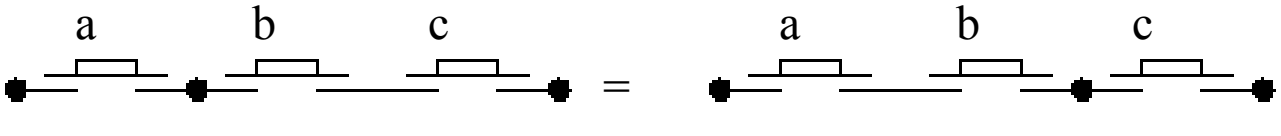
  - **a + b = b + a**

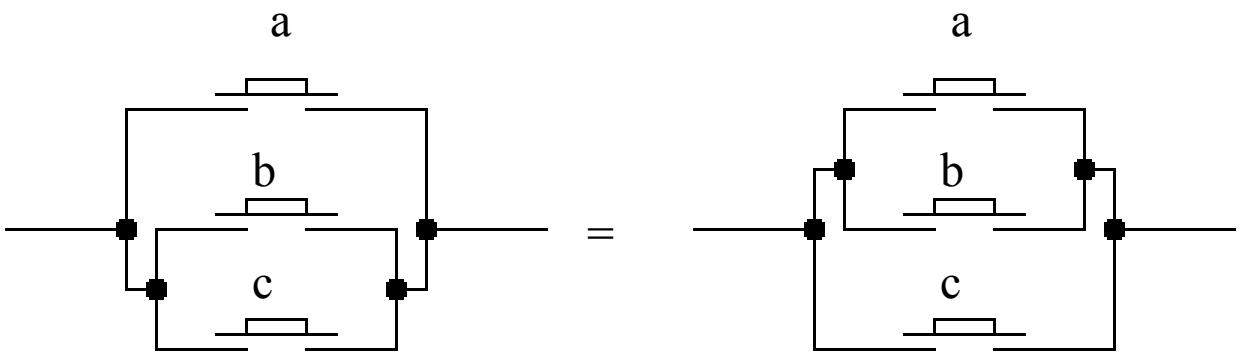


Axiomes de base (2/4)

- **Associativité**
  - **a.(b.c) = (a.b).c**

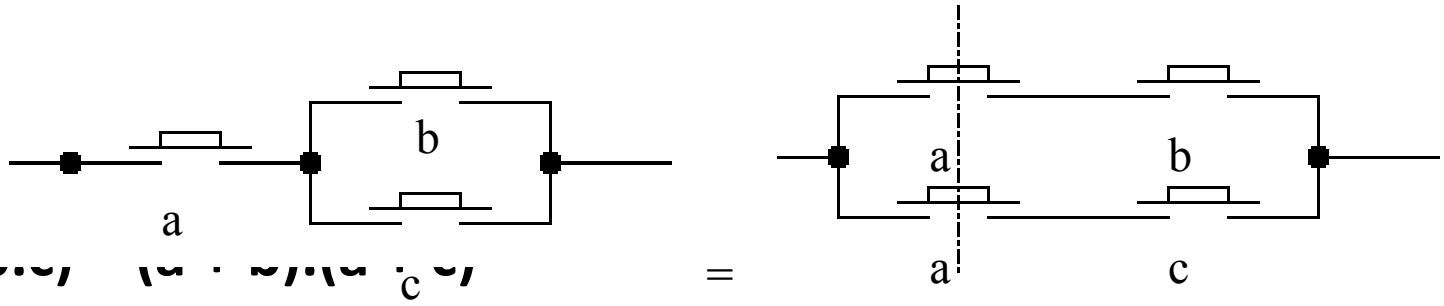
- **a +**

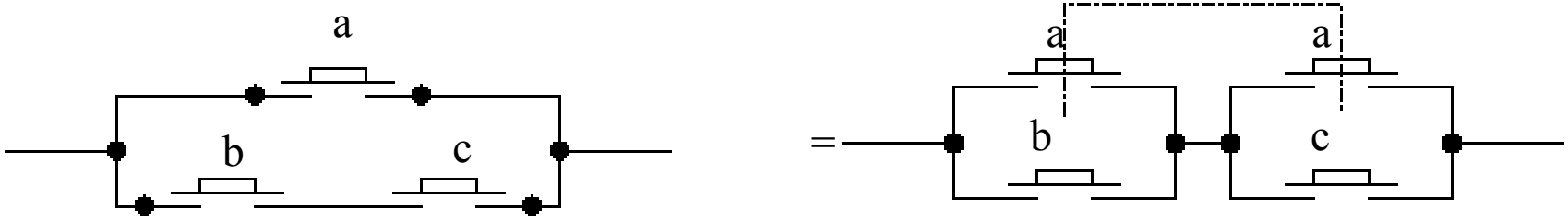




Axiomes de base (3/4)

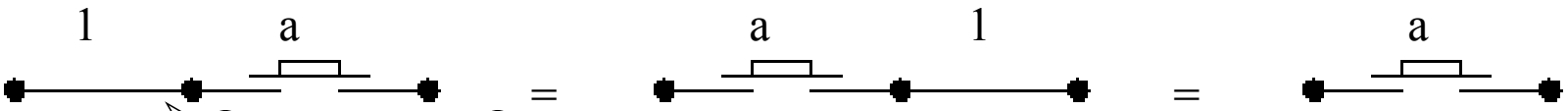
- **Distributivité** :
  - **a.(b + c) = (a.b) + (a.c)**



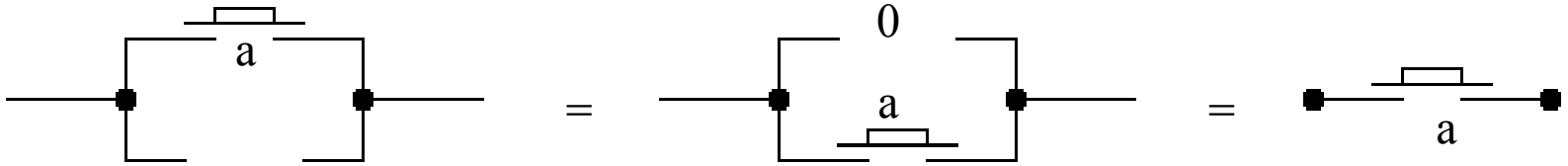


Axiomes de base (4/4)

- **Éléments neutres**
  - **1.a = a.1 = a**



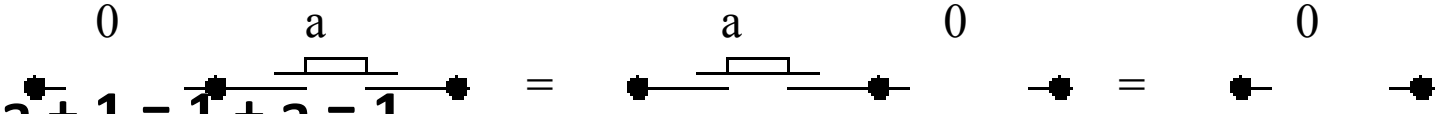
  - **0 + a = a + 0 = a**



- **Complément a.a = 0** **a + a = 1**

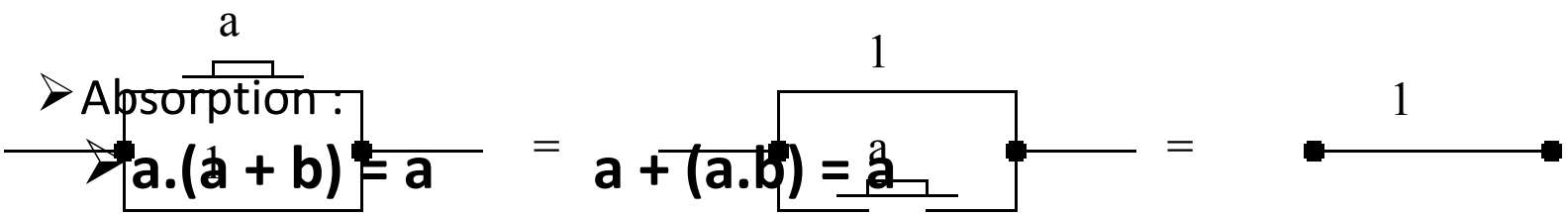
Propriétés (1/2)

- **Élément absorbant :**
  - **a.0 = 0.a = 0**



  - **a + 1 = 1 + a = 1**

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



  - **a + (a.b) = a**

Propriétés (2/2)

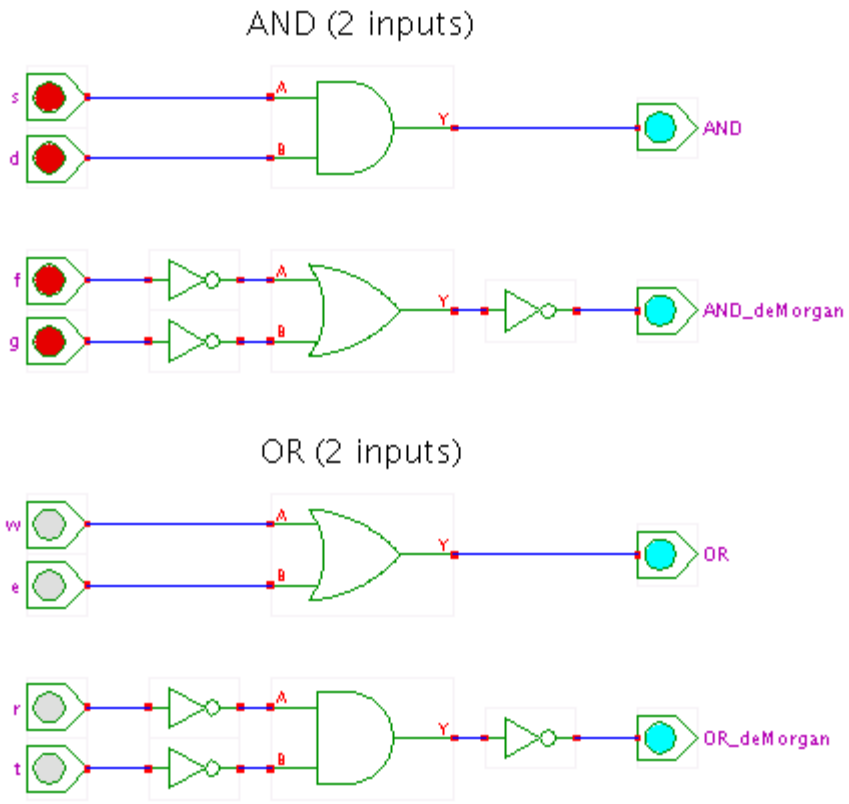
- **Idempotence :**
  - **$a.a = a$ $\quad a + a = a$**
- **Involution :**
  - **$\overline{\overline{a}} = a$**

- **Théorème de De Morgan :**

De Morgan « graphiquement »

Réalisé avec Hades : un outil de simulation gratuit

http://tams-www.informatik.uni-hamburg.de/applets/hades/html/



Un AND3 en VHDL

```
entity AND_3 is
  port(
    e1 : in  bit;
    e2 : in  bit;
    e3 : in  bit;
    s  : out bit
    );
end entity;

architecture arc of AND_3 is

begin  -- arc

  s <= e1 and e2 and e3;

end arc;
```

Synthèse d'un circuit combinatoire

Pour effectuer la synthèse d'un circuit combinatoire, on part de sa table de vérité.
On en extrait les termes des valeurs pour lesquelles la fonction est vraie (1) et on réalise cette fonction en faisant la somme logique de ces termes,
ou encore, on en extrait les termes des valeurs pour lesquelles la fonction est fausse (0) et on réalise cette fonction en faisant le produit logique de ces termes.

Cette réalisation n'est pas toujours optimale. On aura donc la plupart du temps à simplifier les expressions au moyen de l'algèbre booléenne.

Synthèse d'un circuit combinatoire

Exemple : soit la table de vérité suivante :

a	b	c	f	termes
0	0	0	0	
0	0	1	1	$\bar{a}.\bar{b}.c$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	$a.\bar{b}.c$
1	1	0	1	$a.b.\bar{c}$
1	1	1	1	$a.b.c$

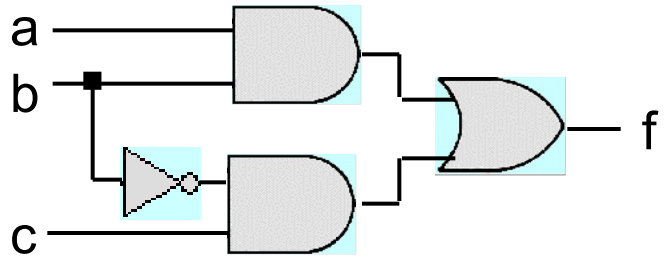
f =?

Synthèse d'un circuit combinatoire

Simplification

f = forme la plus simple possible

Circuit

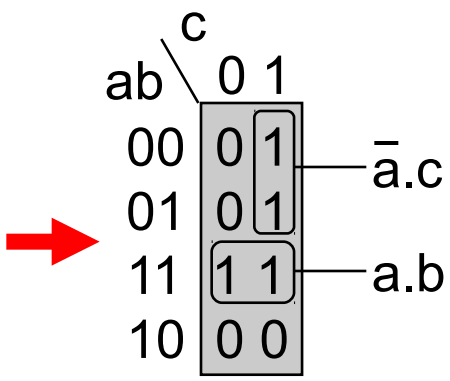


Simplification

La simplification des équations logiques au moyen de l'algèbre booléenne n'est pas toujours simple, et on ne sait pas toujours si on a atteint une solution optimale.
Les tables de Karnaugh permettent de systématiser ce processus.

Tables de Karnaugh

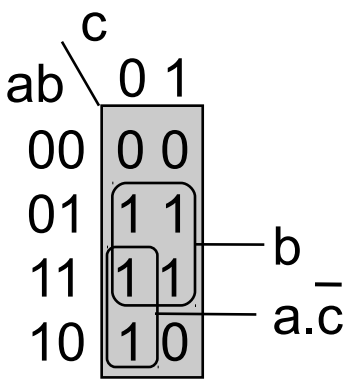
a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

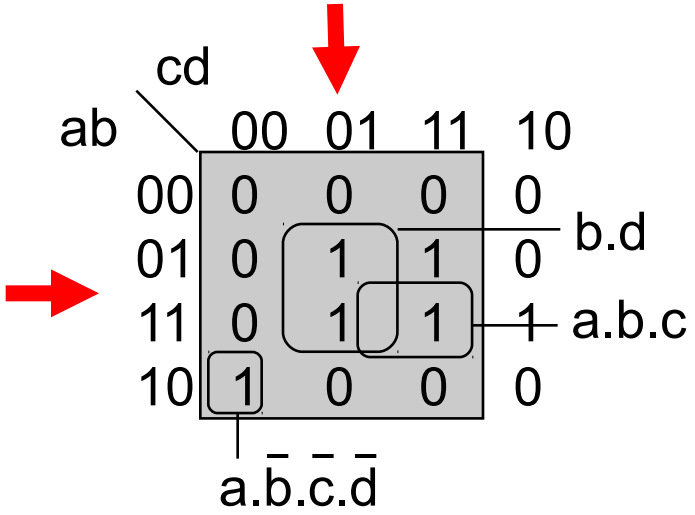


Donc f = a.b + $\bar{a}$.c

Tables de Karnaugh

Chaque boucle doit être rectangulaire et doit contenir le maximum possible de 1 qui soit une puissance de 2 : 1, 2, 4, 8, 16, etc. et ne contenir aucun 0.
La boucle est caractérisée par les combinaisons qui sont vraies pour tous les éléments de la boucle.
Les recouvrements sont possibles.

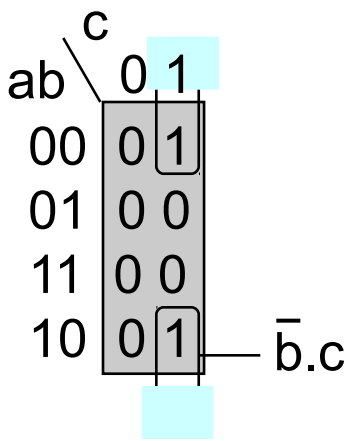




Tables de Karnaugh

Les boucles peuvent «faire le tour» de la table

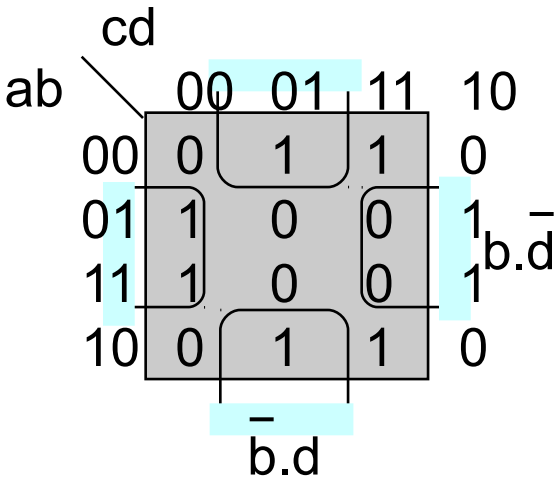
a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

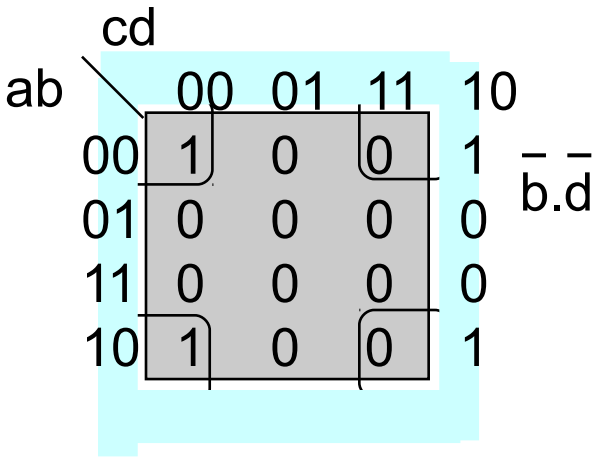


Donc f = $\bar{b}$.c

Tables de Karnaugh

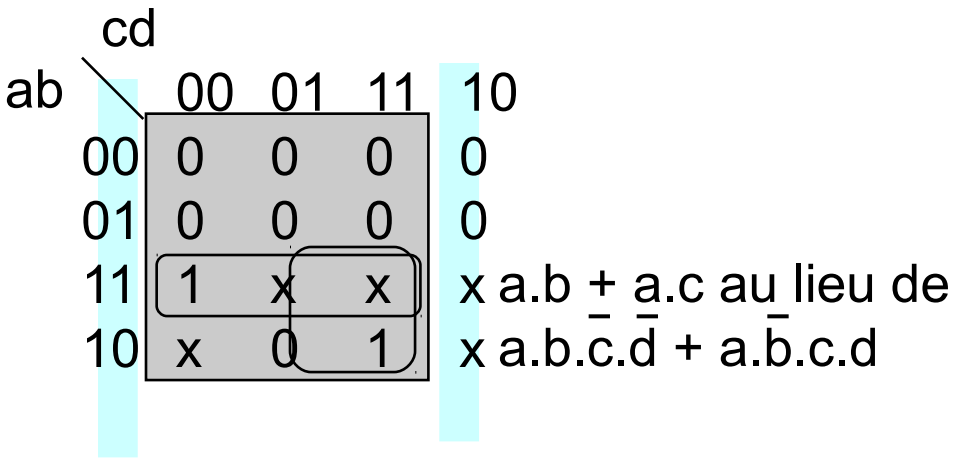
Les boucles peuvent «faire le tour» de la table



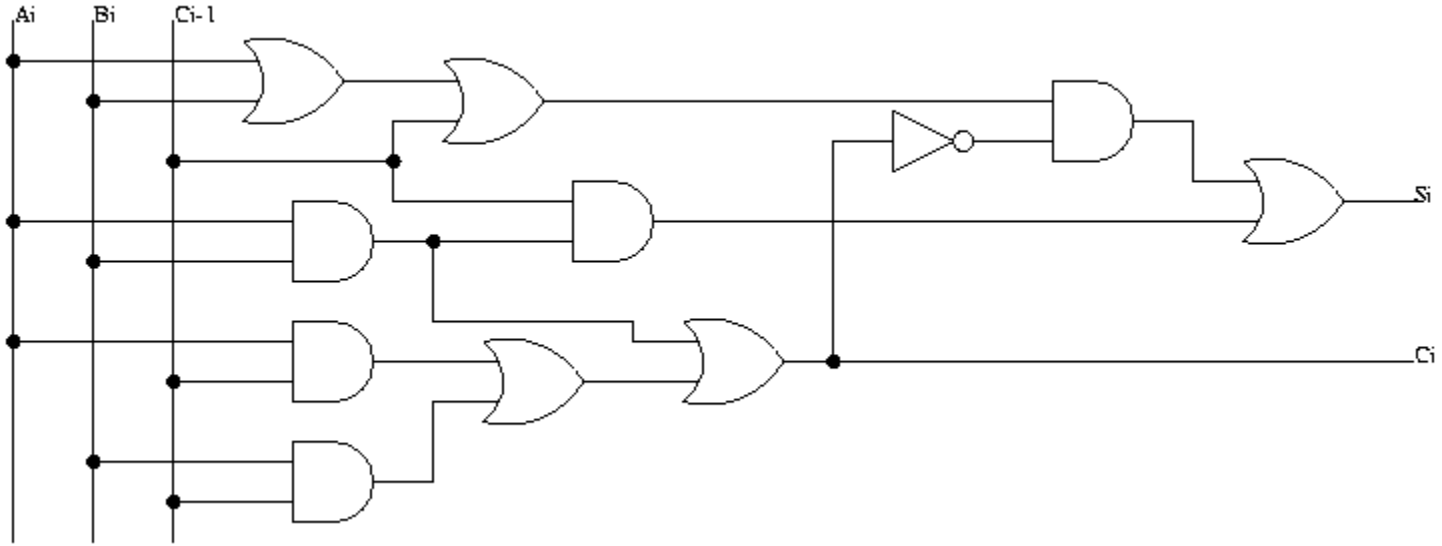


Tables de Karnaugh

Dans certains cas, la sortie pour un état d'entrée donné est indifférente, soit parce que cet état d'entrée ne peut jamais se produire, soit parce que la sortie correspondante ne nous intéresse pas. On inscrit alors un x dans la table de Karnaugh. On peut s'en servir pour minimiser le circuit comme si c'étaient des 1.



Additionneur élémentaire (2/2)



Et en VHDL alors?

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity full_add1 is
  port (
    a, b, cin : in  std_logic;
    s, cout   : out std_logic;
    );
end entity;

architecture arc of full_add1 is

  signal resultat : unsigned(1 downto 0);

begin

  resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
  s        <= resultat(0);
  cout     <= resultat(1);

end arc;
```

A lire

- http://mpicartier.free.fr/ancien_site/electricite/transistor/transit.htm
- Pour un point de vue électronique :
- http://www-lemm.univ-lille1.fr/physique/physicie/lec12.htm
- Un cours VHDL : http://comelec.enst.fr/hdl/vhdl_intro.html