

Architecture des systèmes embarqués SI3

B. Miramond

Objectif du module

- Comprendre les mécanismes matériels essentiels de l'informatique.
- Comprendre l'organisation de l'ordinateur autour de l'élément central, le processeur.
- Pour cela, étudier les différentes couches qui structurent la machine et la philosophie de son utilisation.
- En bref, remonter du circuit numérique jusqu'à l'architecture processeur

En pratique

- Etudier le jeu d'instruction ARM v7
- Réaliser un simulateur de processeur ARM à travers le **projet P-ARM** :
Polytech ARM-based embedded processor
 - *Projet par groupes de 4 étudiants*

Organisation du module

- 7 séances de cours de 1 heure
- 7 séances de TD/projet de 3h
- 2^e semaine de janvier :
 - soutenance des projets par groupes
 - contrôle
- 2 notes : Projet, Contrôle final

Circuits logiques

92

9/9

0800 Antan started
1000 stopped - antan ✓
1300 (032) MP-MC 1.48244000
033 PRO 2 2.130476415
convd 2.130676415
Relays 6-2 in 033 failed special speed test
in relay 11.00 test.

Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multi-Adder Test.

1545

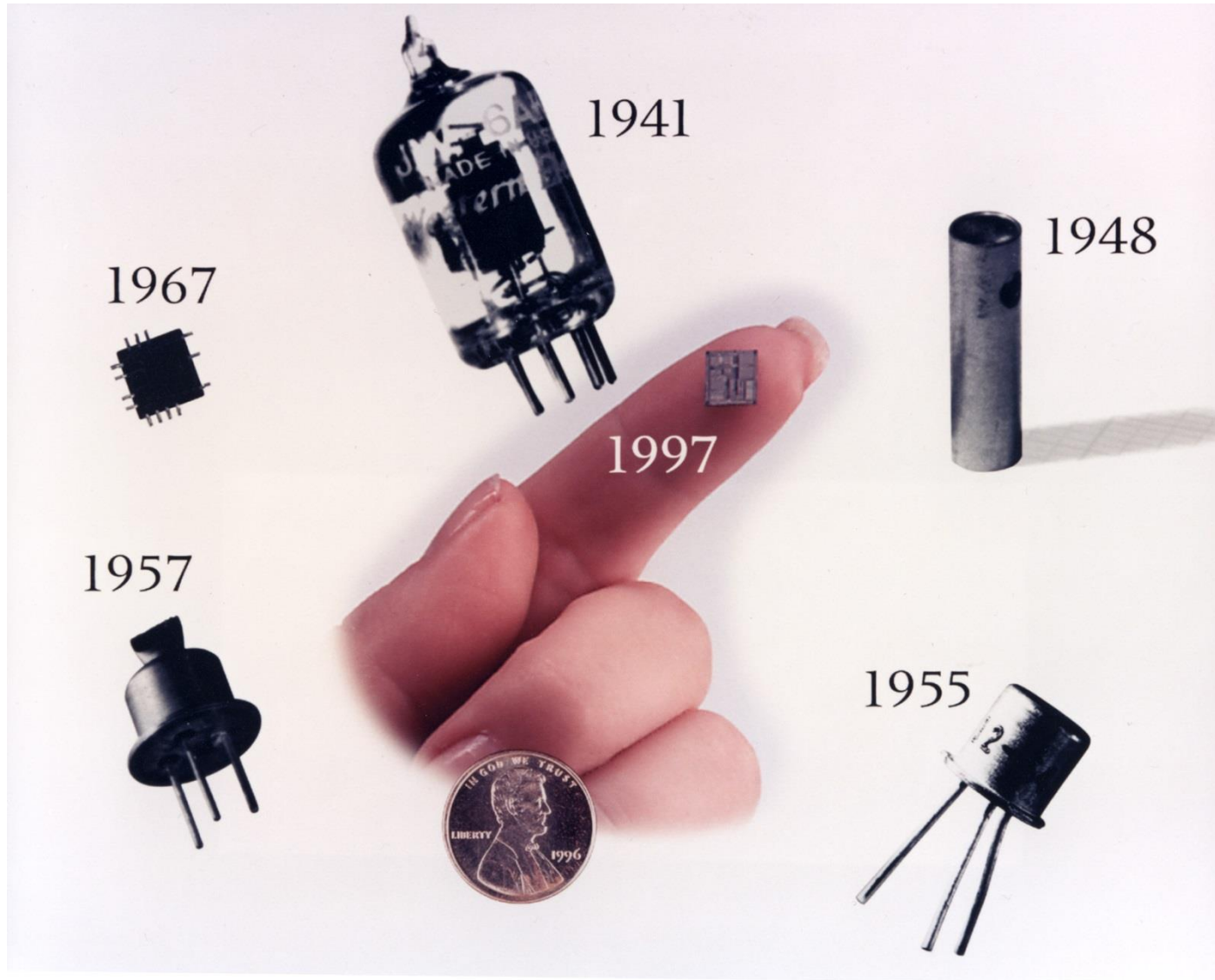


Relay #70 Panel F
(moth) in relay.

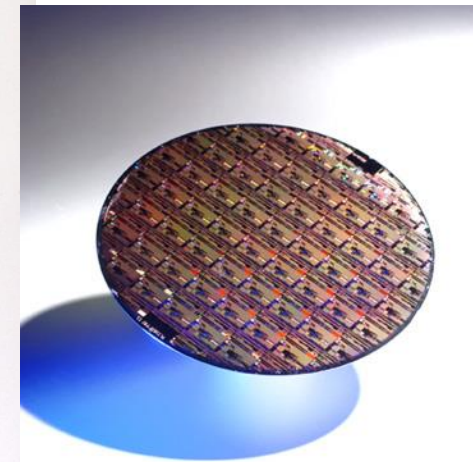
First actual case of bug being found.
1630 antan started.
1700 closed down.

Relay
214.5
Relay 3370

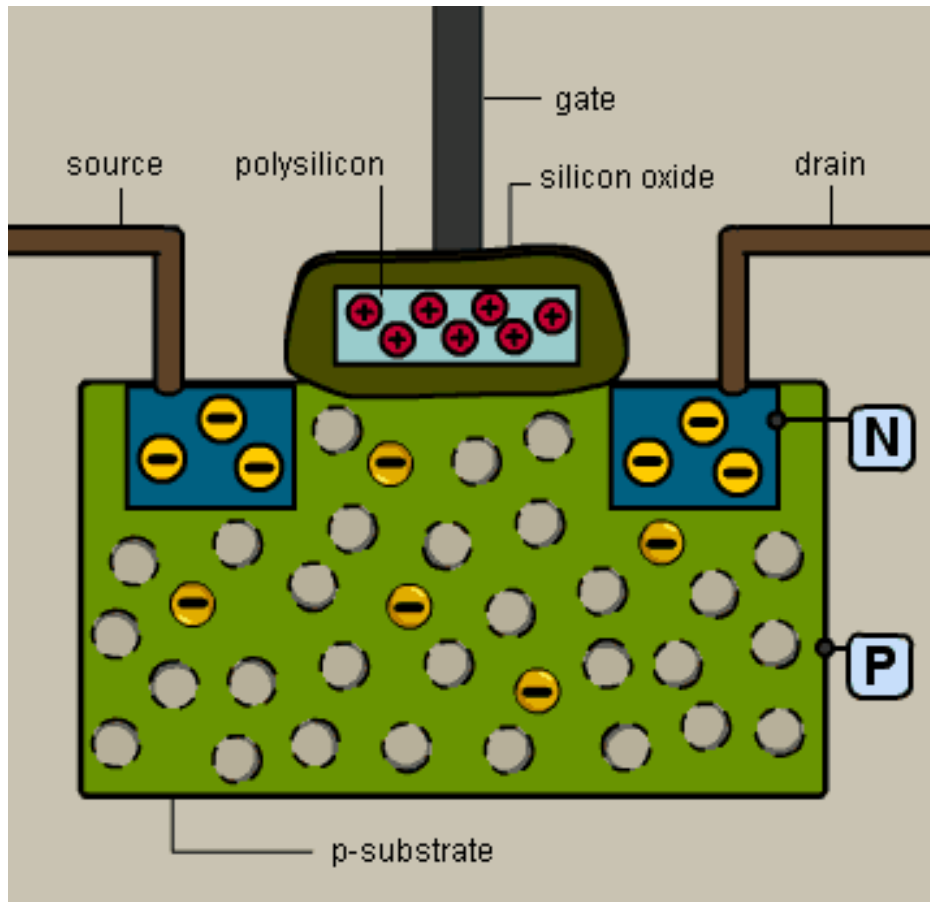
L'historique du transistor



Découvert en 1947
au Bell Labs



Principe de fonctionnement

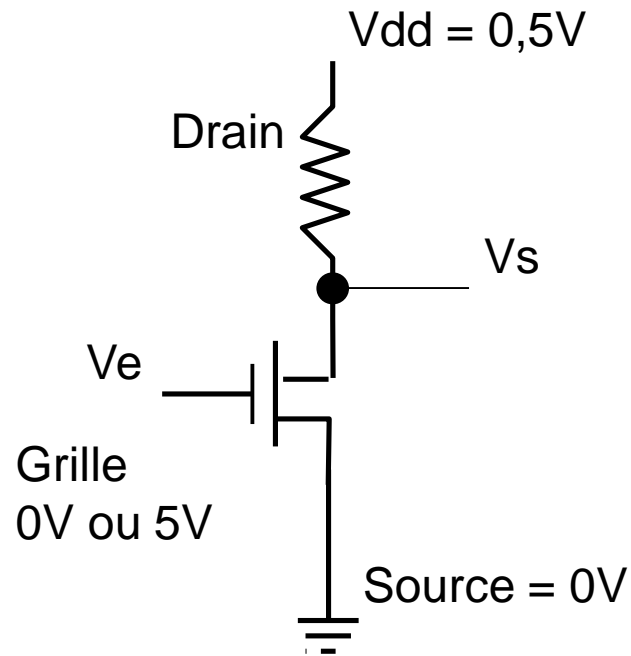


Transistor NMOS dit à canal N

Le transistor à canal P (PMOS)
inverse les polarités

La technologie actuelle utilise des
transistors CMOS qui met en jeu à la
fois des transistors P et N.
Il sont plus rapides et consomment
moins en électricité.

Principe de fonctionnement



- Si $V_e < \text{tension de seuil (0,6V)}$
 - Le transistor est bloquant
 - Interrupteur ouvert ($V_s \cong V_{cc}$)
- Si $V_e > \text{tension de seuil}$
 - Le transistor est passant
 - Interrupteur fermé

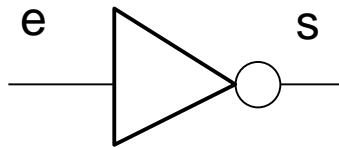
Comportement Inverseur :

V_e	V_s
haut (1)	bas (0)
bas (0)	haut (1)

2. Portes logiques

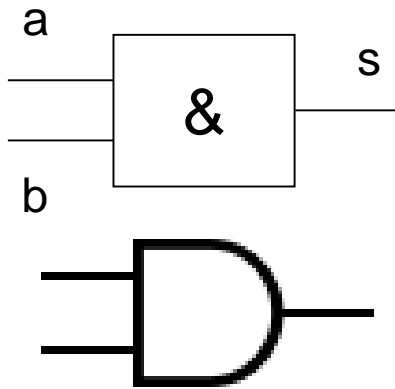
Portes logiques de base

NON

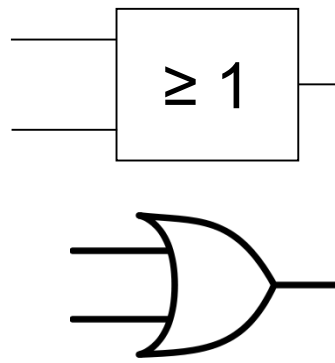


e	s
0	1
1	0

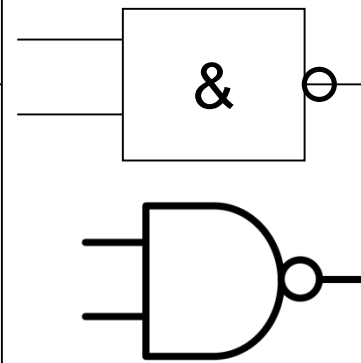
ET



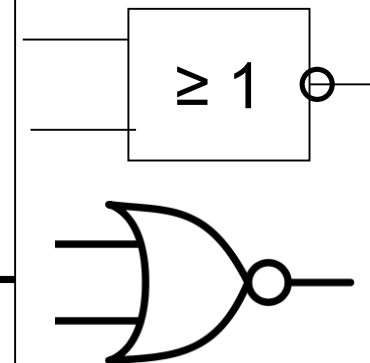
OU



NON-ET

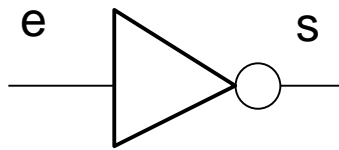


NON-OU



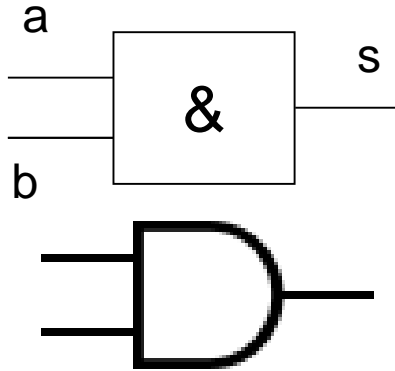
Portes logiques de base

NON



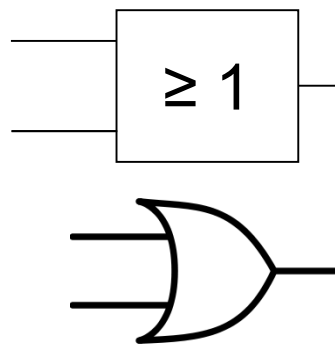
e	s
0	1
1	0

ET

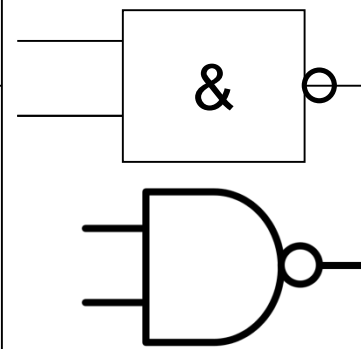


a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

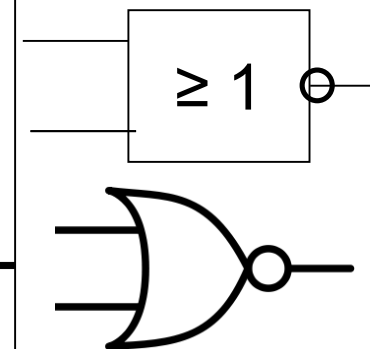
OU



NON-ET

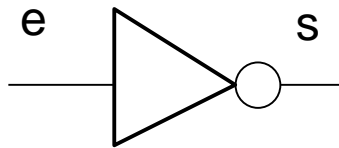


NON-OU



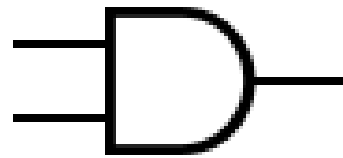
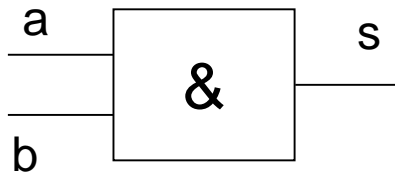
Portes logiques de base

NON



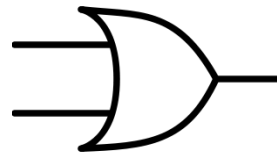
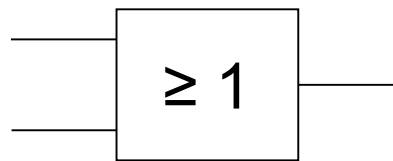
e	s
0	1
1	0

ET



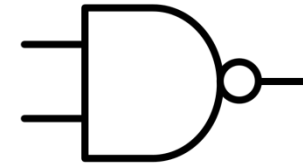
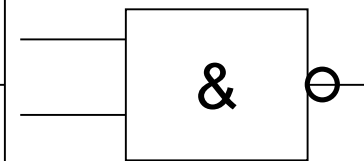
a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

OU

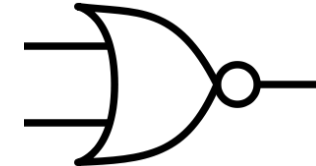
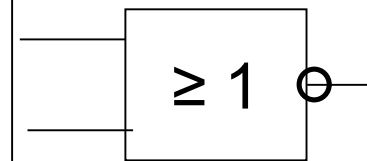


a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

NON-ET

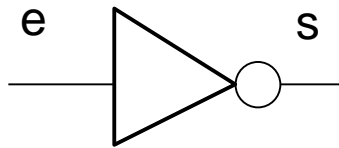


NON-OU



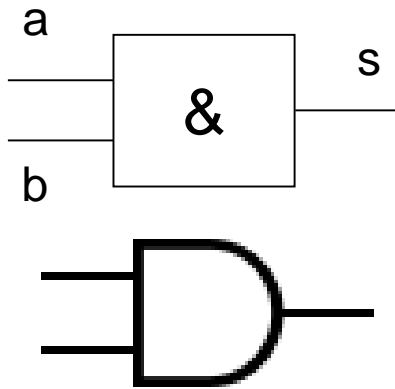
Portes logiques de base

NON



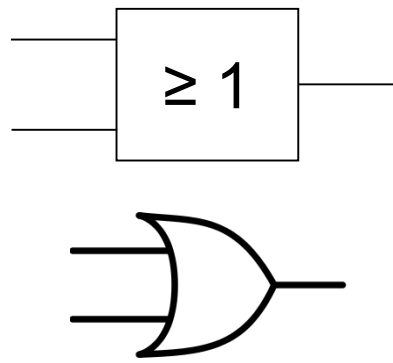
e	s
0	1
1	0

ET



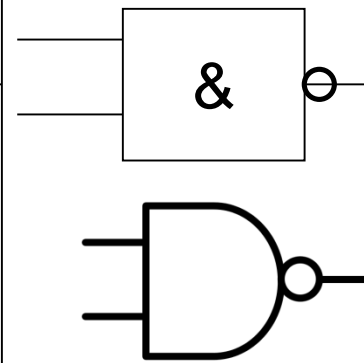
a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

OU



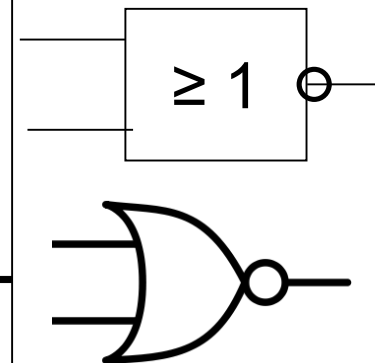
a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

NON-ET



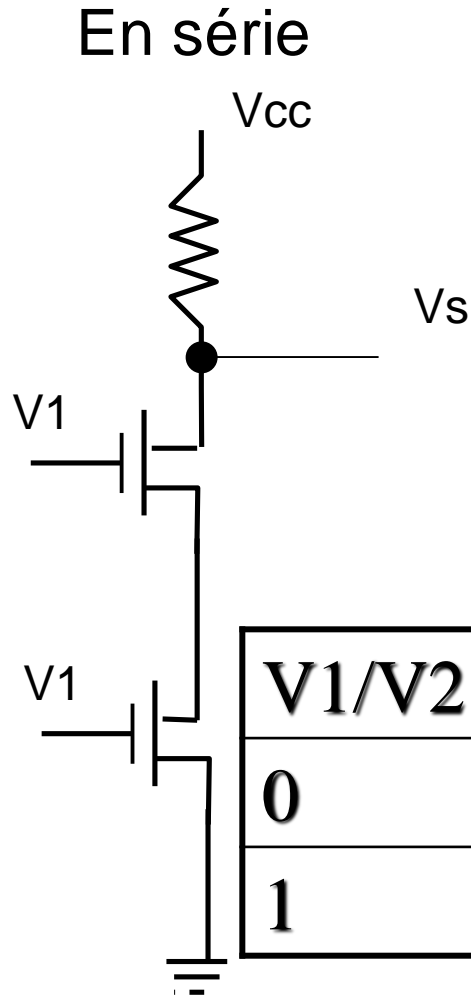
a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

NON-OU

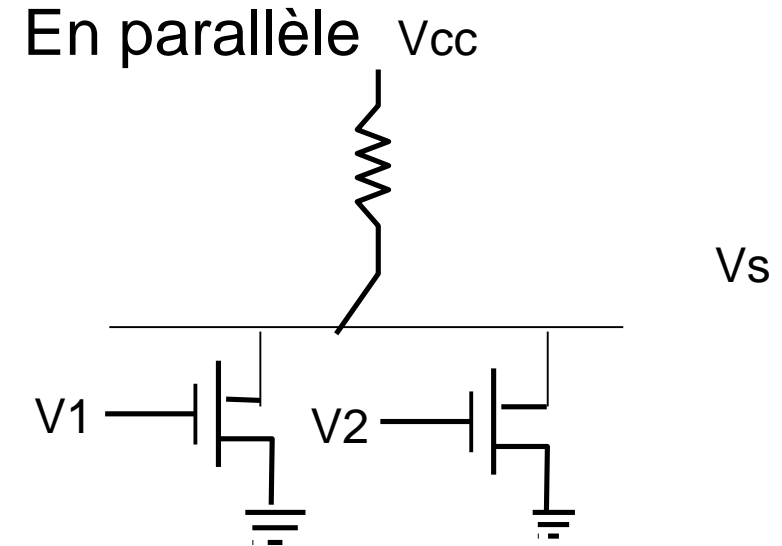


a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

Réalisation de portes en technologie NMOS

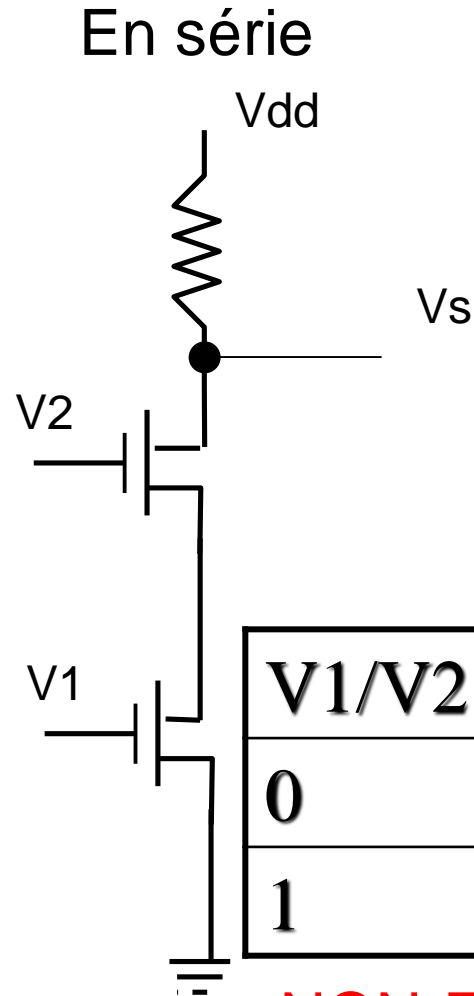


V1/V2	0	1
0	?	?
1	?	?



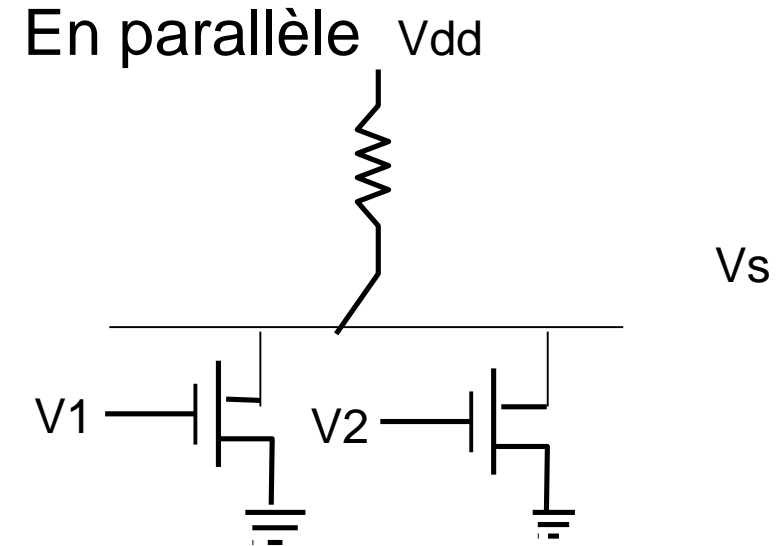
V1/V2	0	1
0	?	?
1	?	?

Réalisation de portes en technologie NMOS



V1/V2	0	1
0	1	1
1	1	0

NON-ET

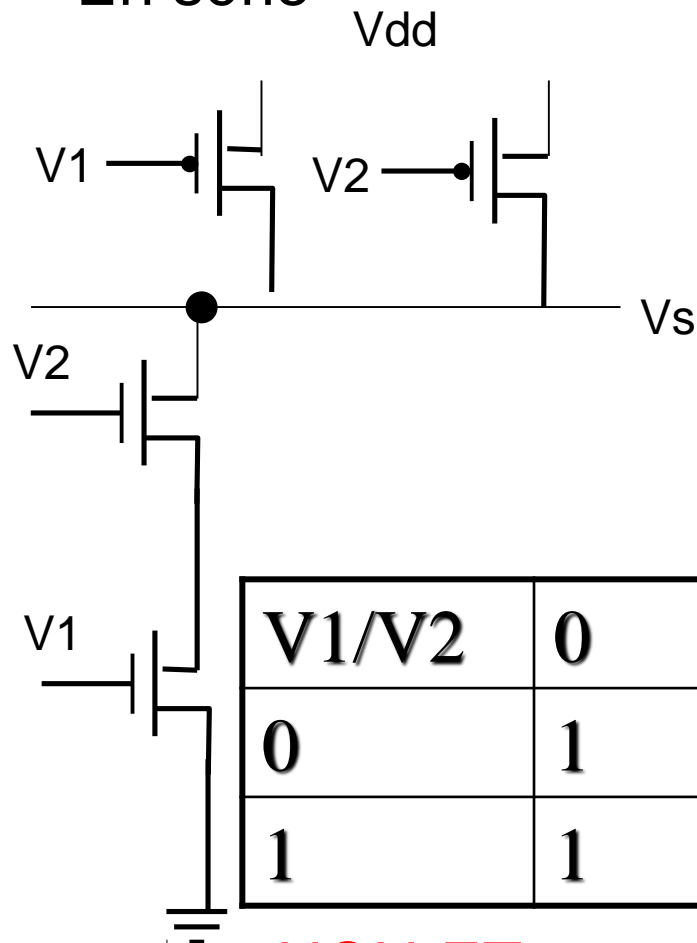


V1/V2	0	1
0	1	0
1	0	0

NON-OU

Réalisation de portes en technologie CMOS

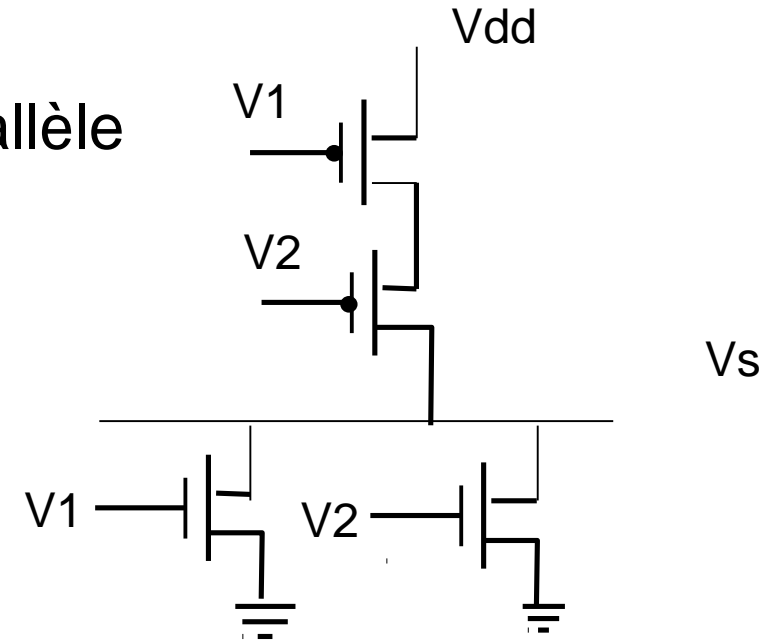
En série



V1/V2	0	1
0	1	1
1	1	0

NON-ET

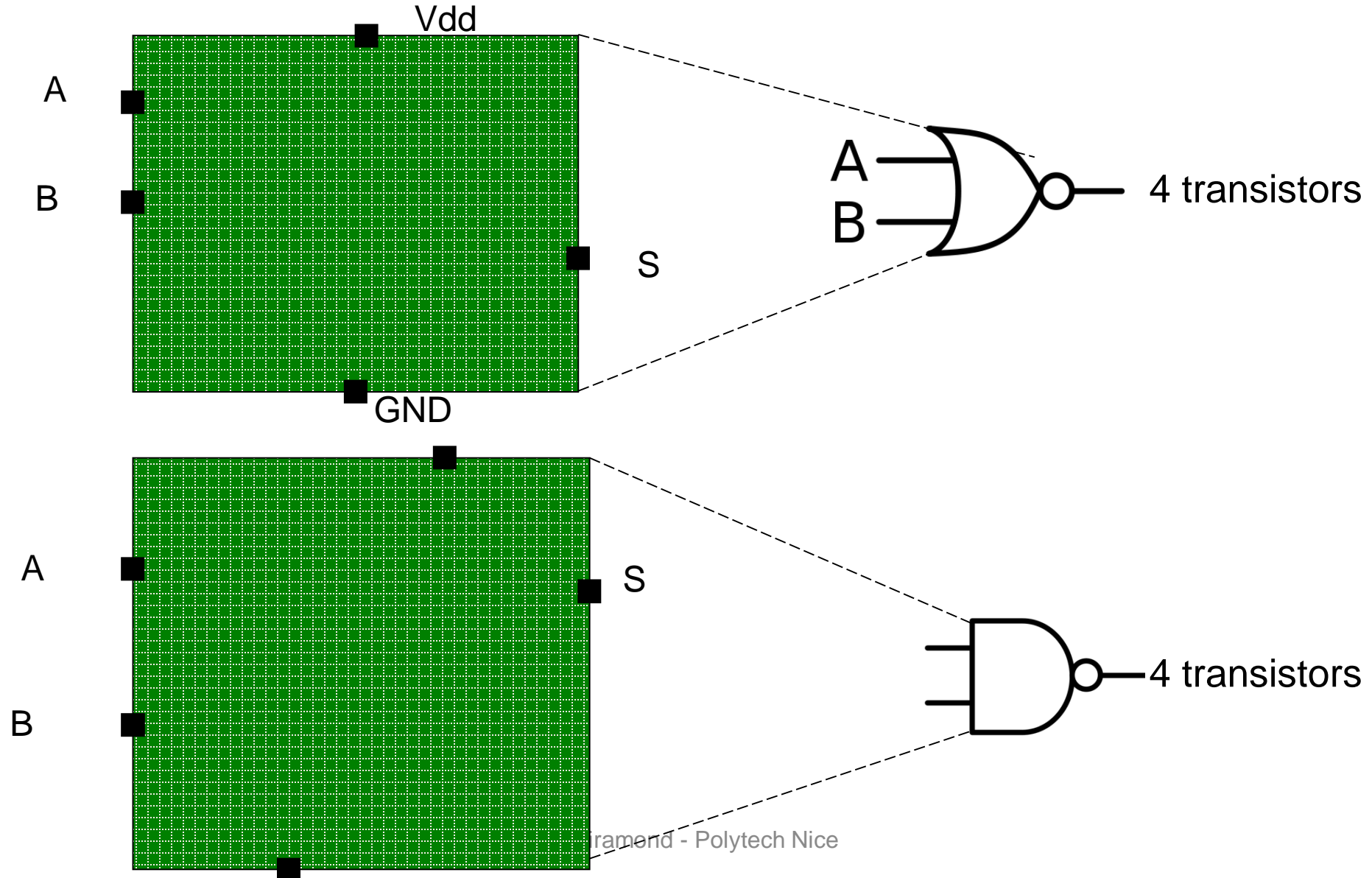
En parallèle



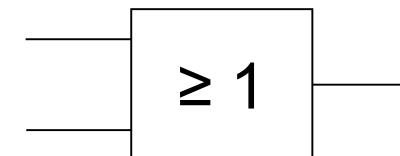
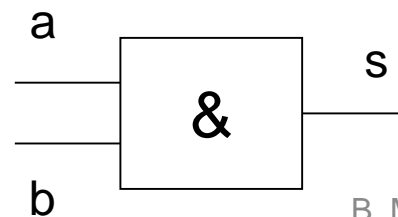
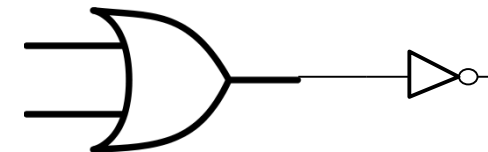
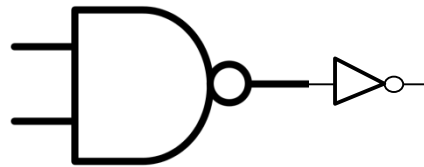
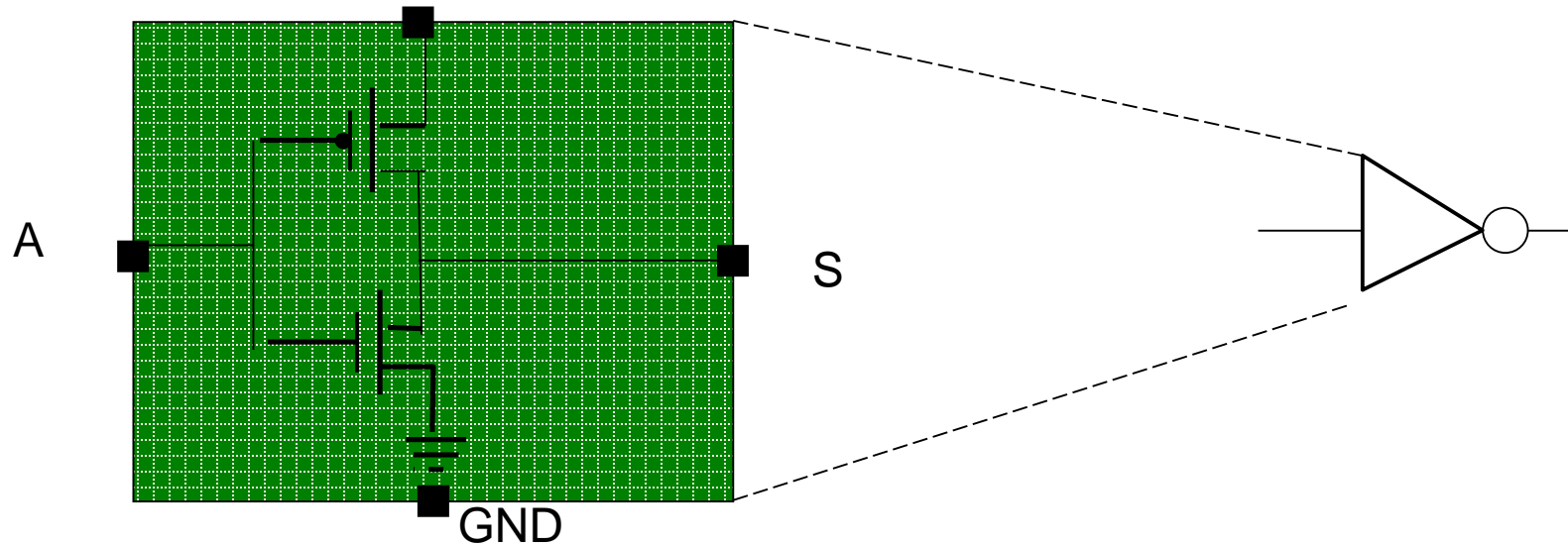
V1/V2	0	1
0	1	0
1	0	0

NON-OU

Portes de bases en technologie CMOS



Portes de bases en technologie CMOS



Réalisation de fonctions logiques

Pour définir chacune des fonctions logiques, on utilise plusieurs **représentations** :

- une représentation électrique : schéma à contacts
- une représentation algébrique : **équation**
- une représentation arithmétique: **table de vérité**
- une représentation temporelle : **chronogramme**
- une représentation logique : **symbole logique**

Fonctions logiques

algèbrique

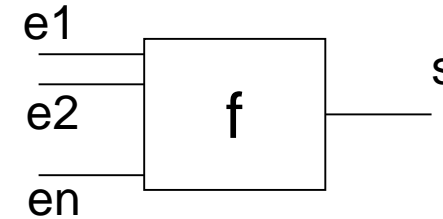
$$s = f(e1, e2, \dots, en)$$

$$B^n \rightarrow B$$

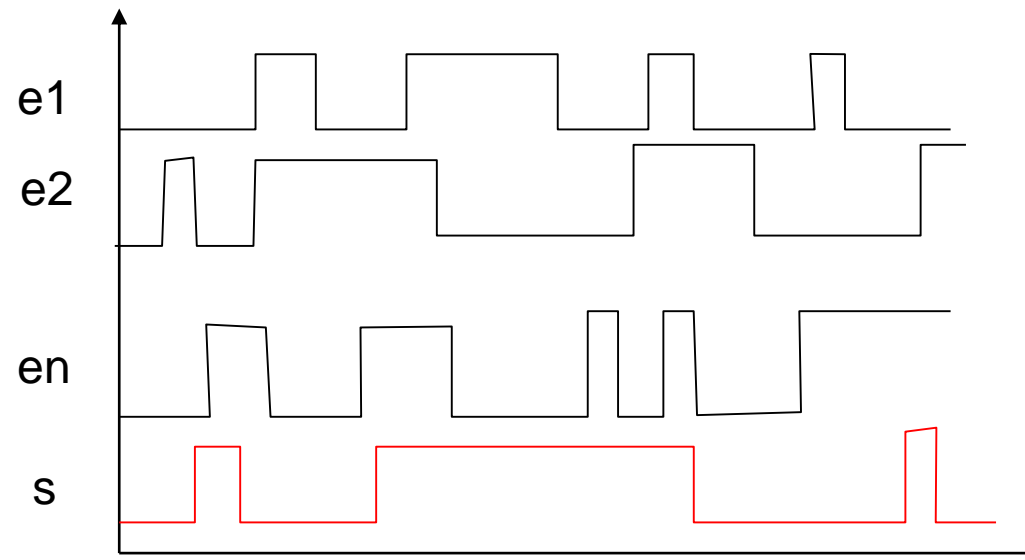
arithmétique

a	b	...	s
0	0		v0
0	1		v1
1	0		v2
...			
1	1		vn

logique



temporelle



Fonctions booléennes

- On peut décrire complètement une fonction booléenne de n variables avec un table de vérité de 2^n lignes

$n=2$ variables

a	b	s
0	0	v0
0	1	v1
1	0	v2
1	1	v3

• 2 valeurs pour v0

• 2 valeurs pour v1

• 2 valeurs pour v2

• 2 valeurs pour v3

$2*2*2*2 = 16$ fonctions
booléennes de deux
variables

Fonctions booléennes

n variables

2 ⁿ lignes	a	b	...	s	
	0	0		v0	• 2 valeurs pour v0
	0	1		v1	• 2 valeurs pour v1
	1	0		v2	• 2 valeurs pour v2
	...				• ...
	1	1		vn	• 2 valeurs pour vn

2^{2ⁿ} fonctions
booléennes de
n variables

Méthode de conception d'un circuit logique

- Etablir la table de vérité
- Simplifier les équations logiques de chaque sortie binaire
- Dessiner le circuit associé à chaque équation
- Connecter le circuit (ici décodeur) à des entrées de test
- Valider en simulation manuelle
- Valider avec des vecteurs de tests

Vecteurs de test

- Les vecteurs de test permettent d'injecter automatiquement des jeux de données et de spécifier la valeur des sorties attendues
- Ils permettent les tests unitaires de chaque composant
- Les vecteurs de test ne sont pas forcément exhaustifs
 - Le format est spécifié dans le TD1

Exemple de circuit – le décodeur 7 segments

- L'afficheur 7 segments permet d'afficher l'ensemble des chiffres hexadécimaux de 0 à F
- 7 segments de contrôle de A à G
- Nécessite un décodeur pour établir la conversion du code hexadécimal sur 4 bits au contrôle de l'afficheur sur 7 bits

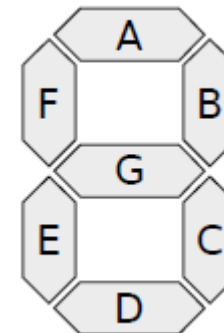
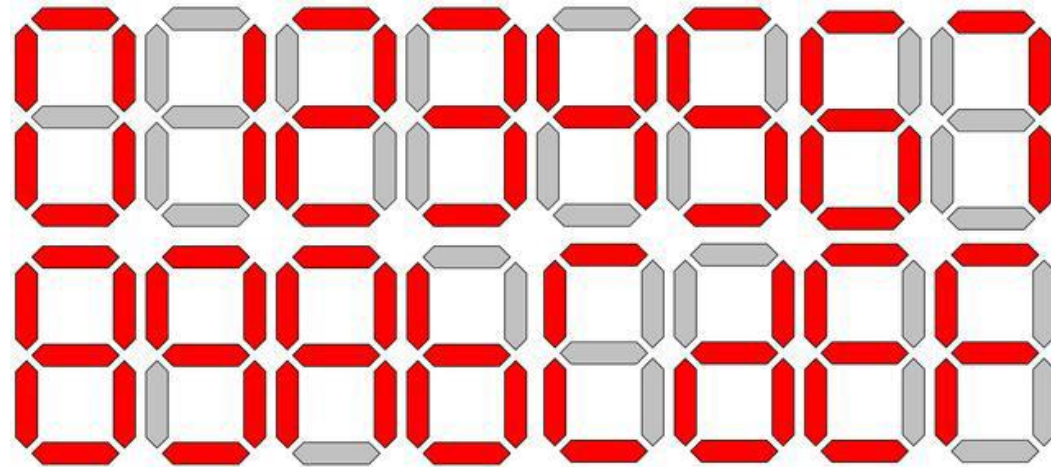



Table de vérité du décodeur

- Elle dispose de 16 lignes
- Les entrées sont les 4 bits codant le chiffre hexa
- Les sorties sont les 7 segments

	Individual Segments						
Display	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2							
3							
4							
5							
6							
7							
8							
9							
A							
B							
C							
D							
E							
F							

Réalisation sous Logisim

Logisim-evolution: decodeur_et_afficheur de decodeur_7_segments_profs (v 2.15.0)

Fichier Editer Projet Simulation FPGAMenu Fenêtre Aide

Placer ici le circuit de décodeur 7 segments
utiliser des tunnels pour connecter
les sorties du décodeur à l'afficheur

Ceci est un compteur. Activer les ticks dans le menu simulation

Et ici un deuxième décodeur

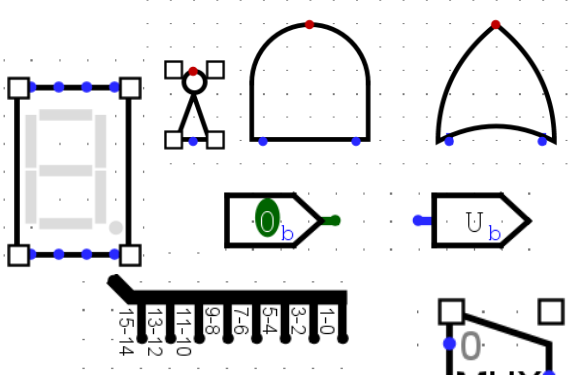
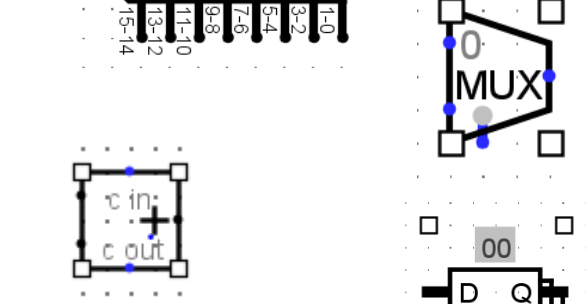
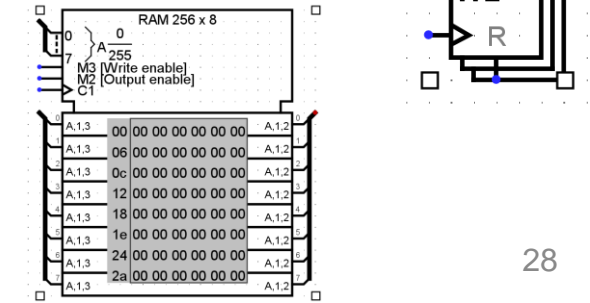
A vous de jouer !

The screenshot shows the Logisim-evolution interface. On the left is a project tree with components like 'main', 'Decodeur_7_seg', 'decodeur_et_afficheur', and 'test'. Below it is a properties panel for the selected component 'Circuit: decodeur_et_afficheur', showing fields for 'Nom du circuit', 'Label partagé', 'Décoration du label', 'Police du label', 'Nouvelle apparence', 'Use fixed box-size', and 'Chemin du fichier d\'...'. The main workspace contains a circuit diagram. It starts with a 'CTR8' counter block, which is connected to a '7-segment decoder' block. The decoder's outputs are connected to two '7-segment display' blocks. The displays show the hexadecimal digits '0' and '1'. Text annotations in French provide instructions: 'Placer ici le circuit de décodeur 7 segments utiliser des tunnels pour connecter les sorties du décodeur à l\'afficheur' (Place the 7-segment decoder circuit here, use tunnels to connect the decoder's outputs to the display), 'Ceci est un compteur. Activer les ticks dans le menu simulation' (This is a counter. Activate the ticks in the simulation menu), and 'Et ici un deuxième décodeur' (And here a second decoder). A red text overlay at the bottom says 'A vous de jouer !' (It's your turn to play!).

Circuit: decodeur_et_afficheur	
VHDL	Verilog
Nom du circuit	decodeur_et_affich...
Label partagé	
Décoration du label ...	Est
Police du label parta...	SansSerif Gras 16
Nouvelle apparence	Oui
Use fixed box-size	Non
Chemin du fichier d'...	

Auto 100%
100%

Liste des principaux composants utilisés durant le projet

Composant	Fonction	Cours correspondant	Symbôle logisim
Portes logiques	logique	CM1	
Afficheur 7 segments	Affichage		
Broches	Ports d'entrée/sortie	-	
Splitter	Découpage/Regroupement d'un bus	-	
Multiplexeur	Aiguillage	CM2	
Décodeur	Décodage binaire	CM2	
Circuits arithmétiques	calculs	CM3	
Registre	Mémoire locale	CM4	
Mémoires (RAM/ROM)	Stockage des données et des instructions	CM6	

Organisation des séances de TD

- 4 groupes de TD encadrés par
 - I. Litovski, A. Muliukov, PE. Novac, B. Miramond
- Constitution des groupes :
 - pour la séance 1, suivre les groupes de HP
 - Pour les autres séances, selon les groupes de projet : 4 étudiants par groupe
 - Inscription depuis le lien sur moodle : <http://sondages.polytech.unice.fr/index.php/188847?lang=fr>
- Pour la première séance :
 - Introduction à Logisim utilisé pendant le reste du module
 - Suivre le sujet, répondez aux questions et seulement après avoir essayer poser vos questions aux chargés de TD
- Pour les autres séances
 - Réalisation du projet étape par étape (cf. consignes au cours No 2)
 - Suivez les consignes du chargé de TD et organisez vous en groupe

SIMPLIFICATION DE FONCTIONS LOGIQUES

Rappels –

Equivalence et simplifications de fonctions

- On démontre que toute fonction logique peut se décrire à l'aide des trois opérations de base grâce au théorème de De Morgan
- OU
- ET
- NOT

Théorème de De Morgan

- $\overline{a + b} = \bar{a} \bar{b}$
 - Dans les deux cas, l'expression ne sera VRAIE que si a et b sont fausses.
- $\overline{a.b} = \bar{a} + \bar{b}$
 - Dans les deux cas, l'expression ne sera VRAIE que si a ou b sont fausses

Propriétés de la logique

Associativité

- Certaines parenthèses sont inutiles:
 $(a + b) + c = a + (b + c) = a + b + c$
 $(a \cdot b) \cdot c = a \cdot (b \cdot c) = a \cdot b \cdot c$

Commutativité

- L'ordre est sans importance.
 $a + b = b + a$
 $a \cdot b = b \cdot a$

Distributivité

- Comme avec les opérations habituelles, il est possible de distribuer:
 $a \cdot (b + c) = a \cdot b + a \cdot c$

Idempotence

- $a + a + a \dots = a$
 $a \cdot a \cdot a \dots = a$

Propriétés

	Forme OU	Forme ET
• Loi d'identité	• $a+0 = a$	• $a.1 = a$
• Loi de nullité	• $a+1 = 1$	• $a.0 = 0$
• Loi d'idempotence	• $a+a = a$	• $a.a = a$
• Loi d'inversion	• $a+\overline{a} = 1$	• $a.\overline{a} = 0$
• Loi d'absorption	• $a+a.b = a$	• $a.(a+b) = a$

TABLES DE KARNAUGH

Réduction d'expression logique

Table de Karnaugh

- La table de karnaugh est une représentation différente de la table de vérité utilisant un codage de Gray (changement d'un seul bit entre chaque configuration binaire).
- Une fois remplie, la réduction consiste à regrouper les cases dont la valeur est '1' par puissance de 2 : 1, 2, 4, 8, ou 16 cases.
- Pour obtenir la fonction la plus réduite, il faut minimiser le nombre de regroupement.

Structure et construction de la table (exemple d'une fonction à 3 variables)

A\BC	00	01	11	10
0				
1				

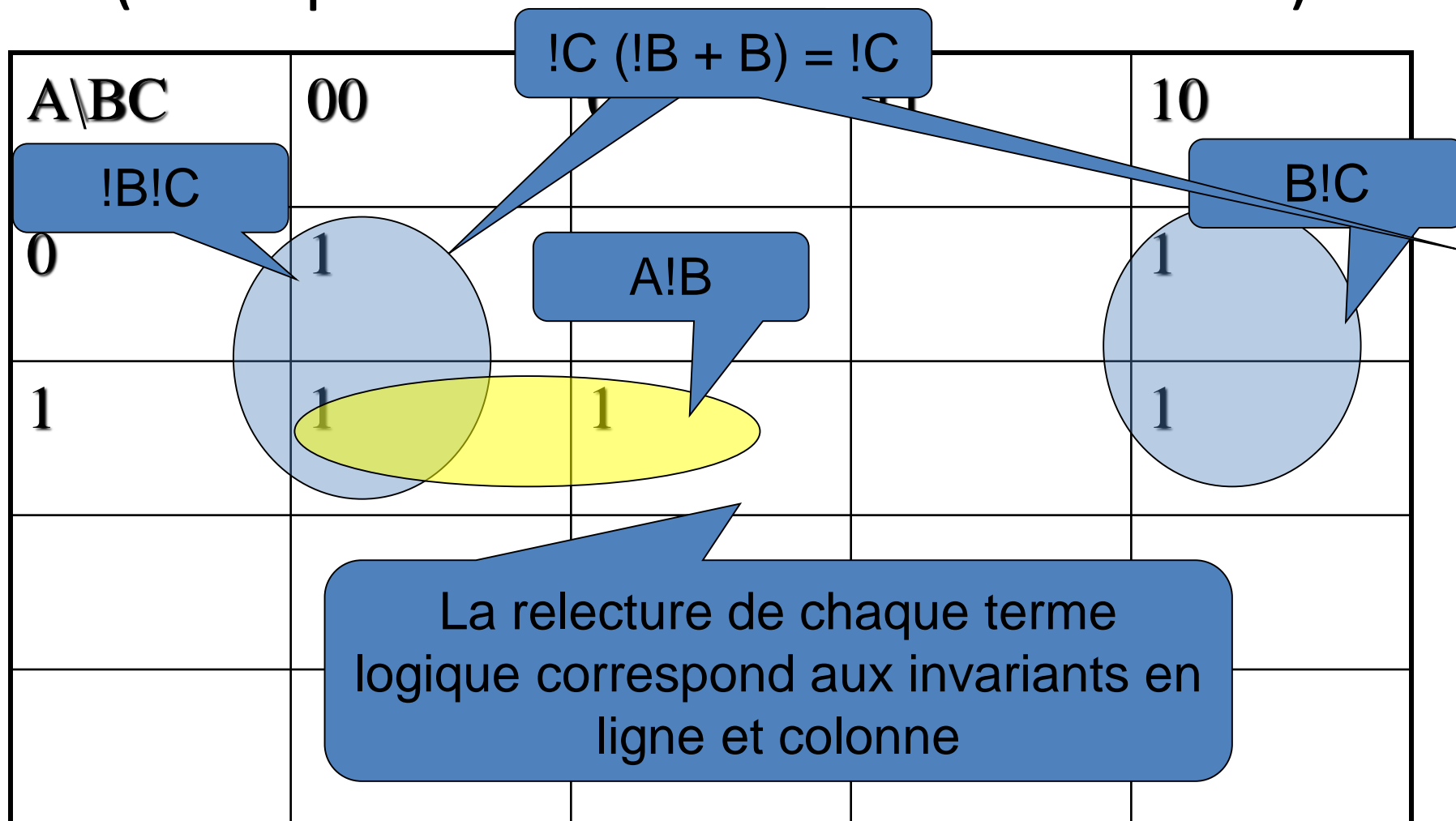
Dans la construction de la table, les valeurs des variables ne doivent changer qu'une par une entre colonnes et lignes (Gray code) !!

Structure et construction de la table (exemple d'une fonction à 3 variables)

A\BC	00	01	11	10
0	1			1
1	1	1		1

Le remplissage de la table suit celui de la table de vérité

Structure et construction de la table (exemple d'une fonction à 3 variables)



Structure et construction de la table (exemple d'une fonction à 3 variables)

A\BC	00	01	11	10
0	1			1
1	1	1		1
$F = !C + A!B$				

Réduction d'expression logique

Table de Karnaugh

- Réduction de fonction à 2 variables
 - Les regroupements d'une seule case correspondent à un terme à 2 variables
 - Les regroupements de 2 cases correspondent à un terme à 1 variable
 - Les regroupements de 4 cases correspondent à un terme à 0 variable
: $f = 1$

Réduction d'expression logique

Table de Karnaugh

- Réduction de fonction à 3 variables
 - Les regroupements d'une seule case correspondent à un terme à 3 variables
 - Les regroupements de 2 cases correspondent à un terme à 2 variables
 - Les regroupements de 4 cases correspondent à un terme à 1 variable
 - Les regroupements de 8 cases correspondent à un terme à 0 variable : $f = 1$

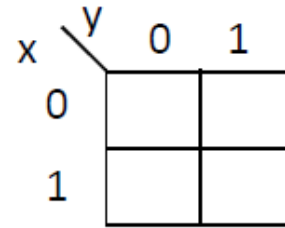
Réduction d'expression logique

Table de Karnaugh

- Réduction de fonction à 4 variables
 - Les regroupements d'une seule case correspondent à un terme à 4 variables
 - Les regroupements de 2 cases correspondent à un terme à 3 variables
 - Les regroupements de 4 cases correspondent à un terme à 2 variables
 - Les regroupements de 8 cases correspondent à un terme à 1 variable
 - Les regroupements de 16 cases correspondent à un terme à 0 variable : $f = 1$

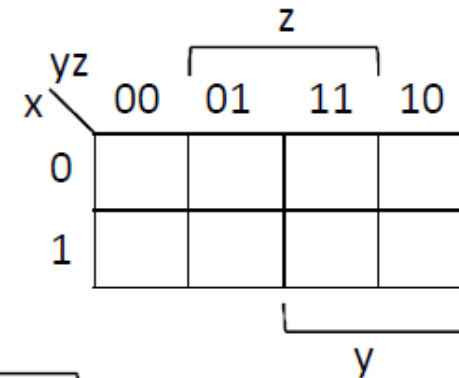
Tables de Karnaugh

Two-Variable K-Maps

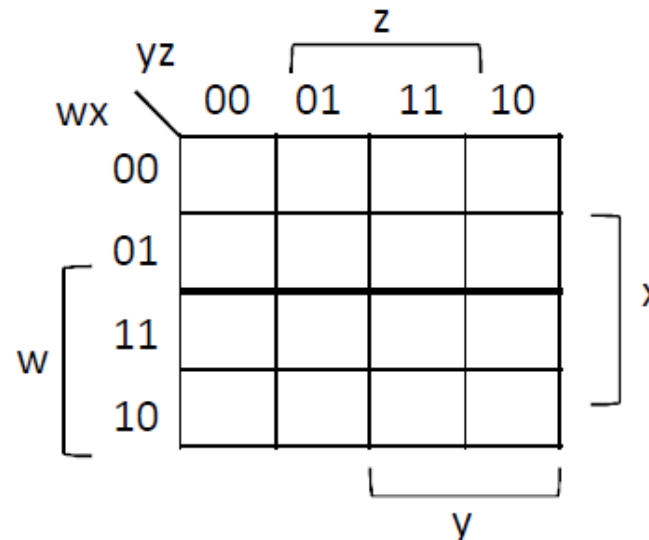


Adjacency
wrap-around
Manhattan
topology

Three-Variable K-Maps



Four-Variable K-Maps

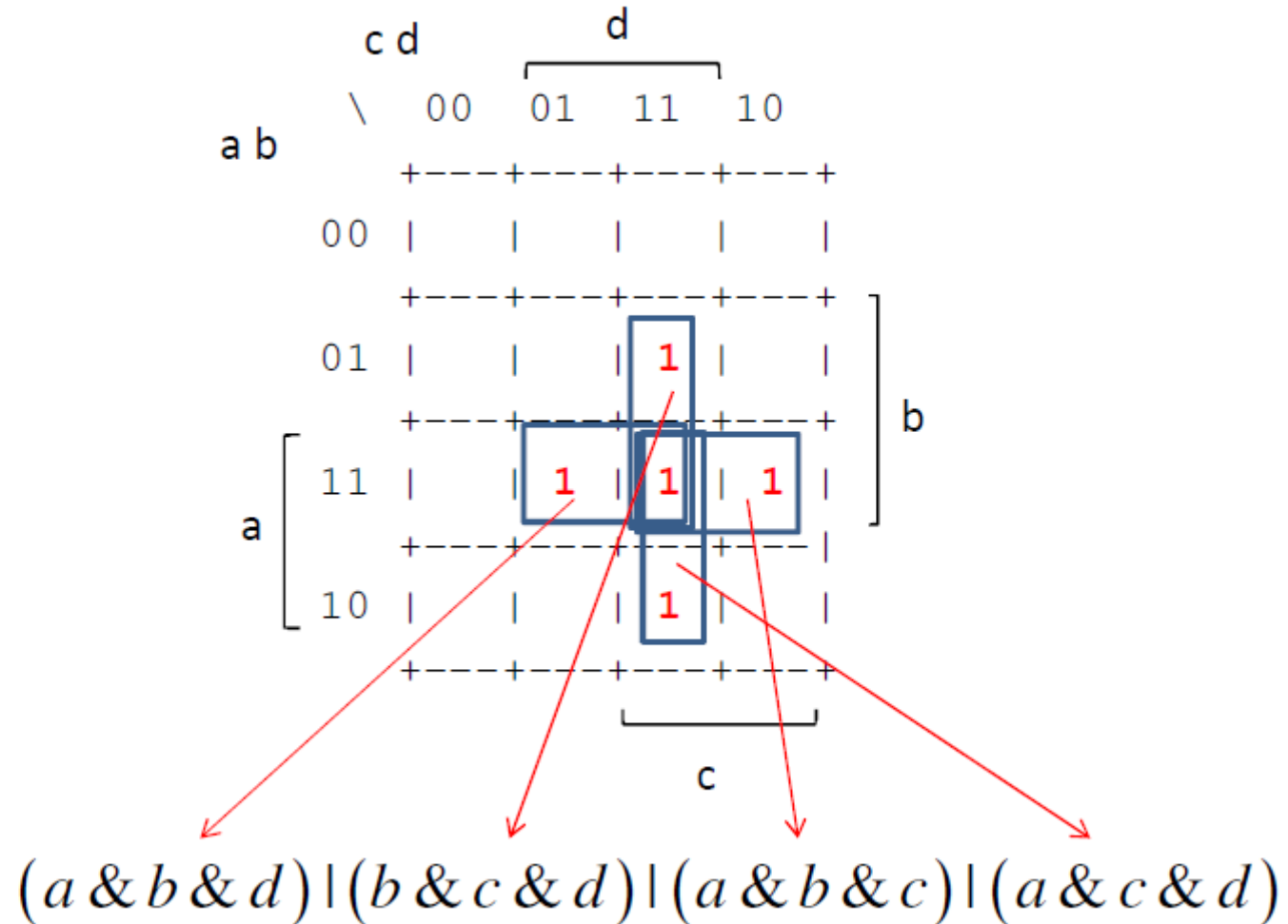


Exemple de K-Map

Circuit de Majorité 4-bits

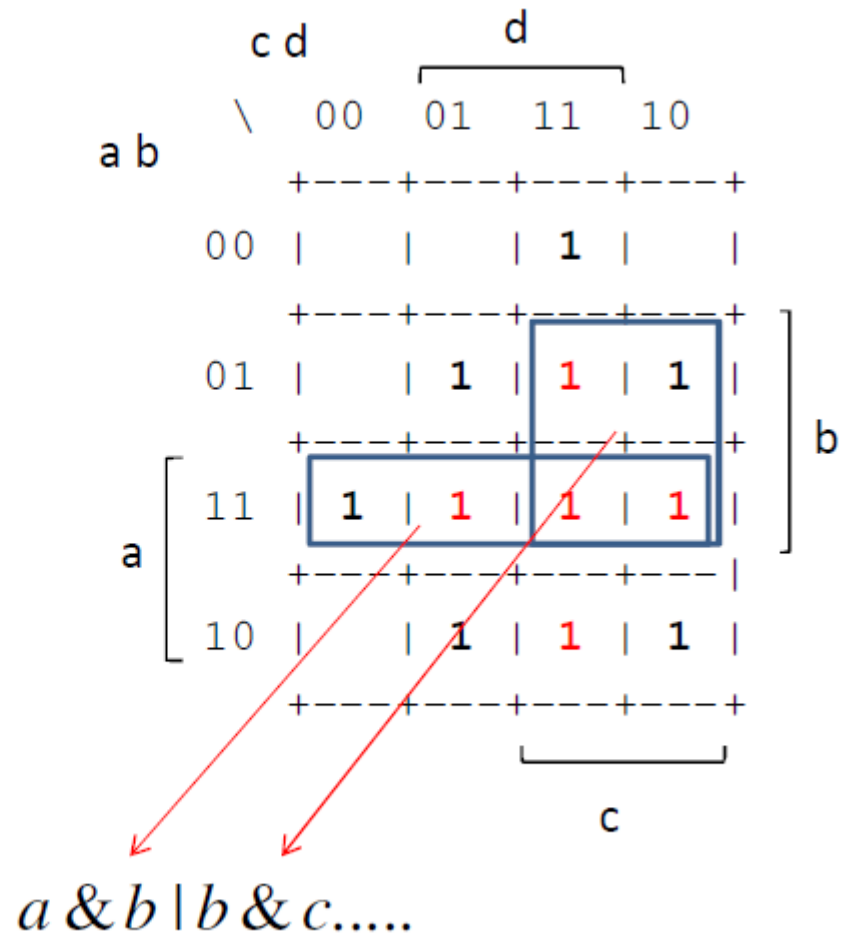
(plus de 1 que de 0)

a	b	c	d	Maj
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



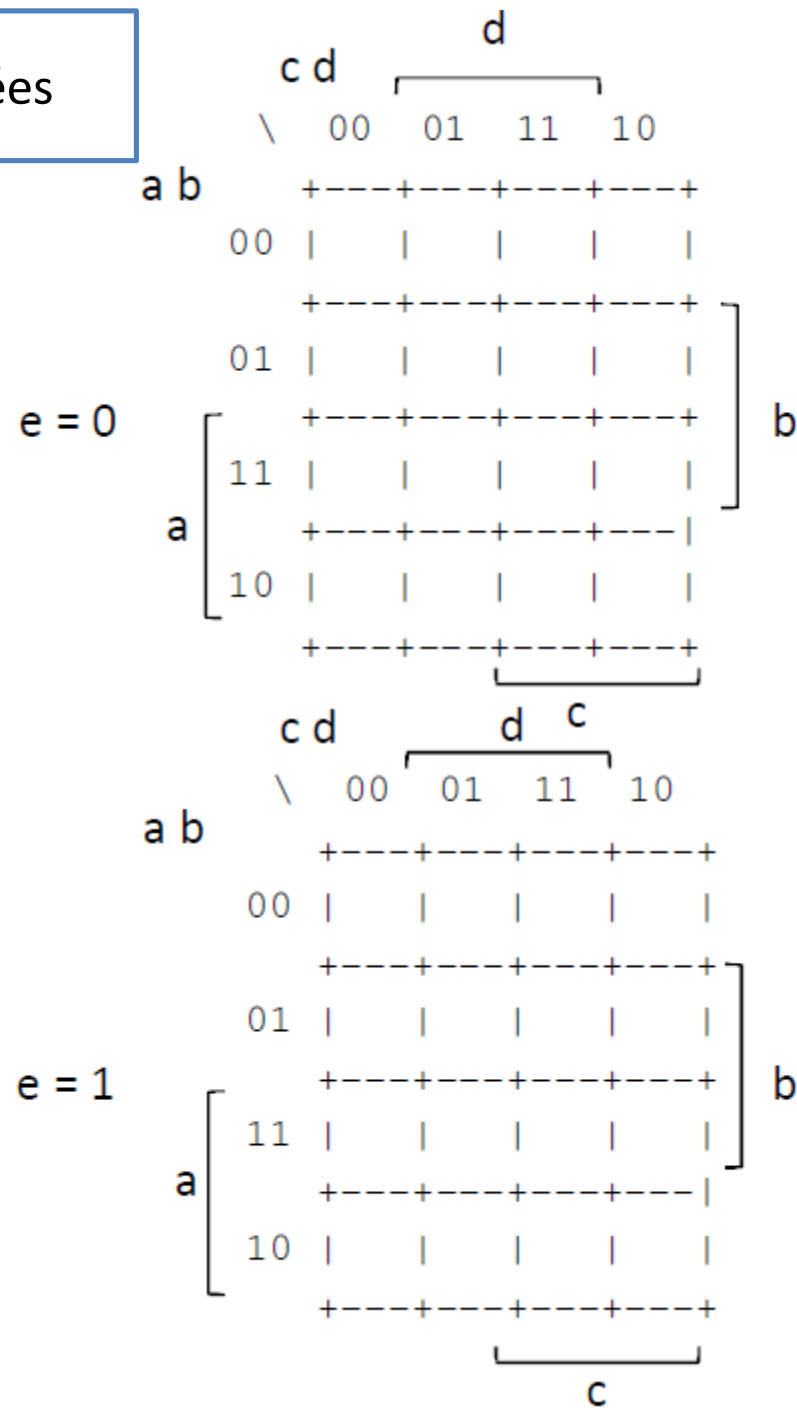
Fonction « plus grand ou égal » (GTE Greater Than or Equal)

a	b	c	d		GTE
0	0	0	0		0
0	0	0	1		0
0	0	1	0		0
0	0	1	1		1
0	1	0	0		0
0	1	0	1		1
0	1	1	0		1
0	1	1	1		1
1	0	0	0		0
1	0	0	1		1
1	0	1	0		1
1	0	1	1		1
1	1	0	0		1
1	1	0	1		1
1	1	1	0		1
1	1	1	1		1



Exemple de circuit Majorité à 5 entrées

a	b	c	d	e	maj	a	b	c	d	e	maj
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	0	0	1	0	0	1	0	0
0	0	0	1	1	0	1	0	0	1	1	1
0	0	1	0	0	0	1	0	1	0	0	0
0	0	1	0	1	0	1	0	1	0	1	1
0	0	1	1	0	0	1	0	1	1	0	1
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	0	0	1	1	0	0	0	0
0	1	0	0	1	0	1	1	0	0	1	1
0	1	0	1	0	0	1	1	0	1	0	1
0	1	0	1	1	1	1	1	0	1	1	1
0	1	1	0	0	0	1	1	1	0	0	1
0	1	1	0	1	1	1	1	1	0	1	1
0	1	1	1	0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1



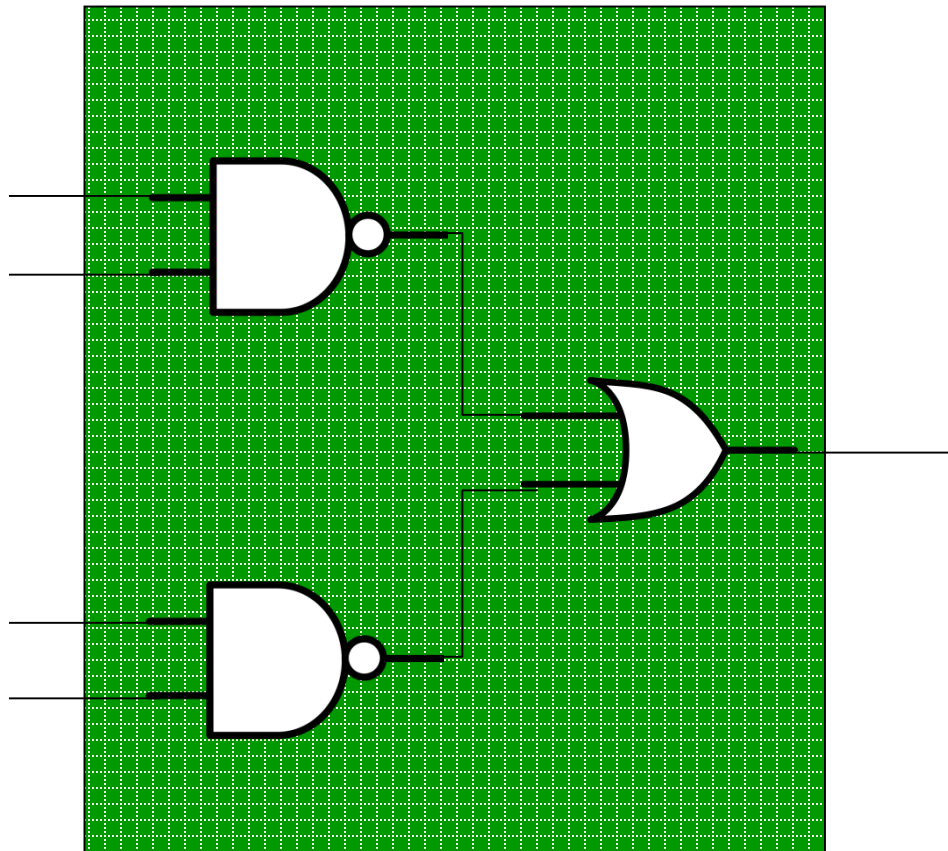
Simplification d'expressions logiques

- Il n'est pas toujours évident de savoir si on a atteint l'expression logique minimale
- La méthode de K-Maps le permet mais pour un nombre réduit de variables (4 ou 5)
- L'algorithme de Quine-Mac Cluskey est une méthode
 - systématique fonctionnant quelque soit le nombre de variables logiques
 - et pouvant être programmée

Bonus

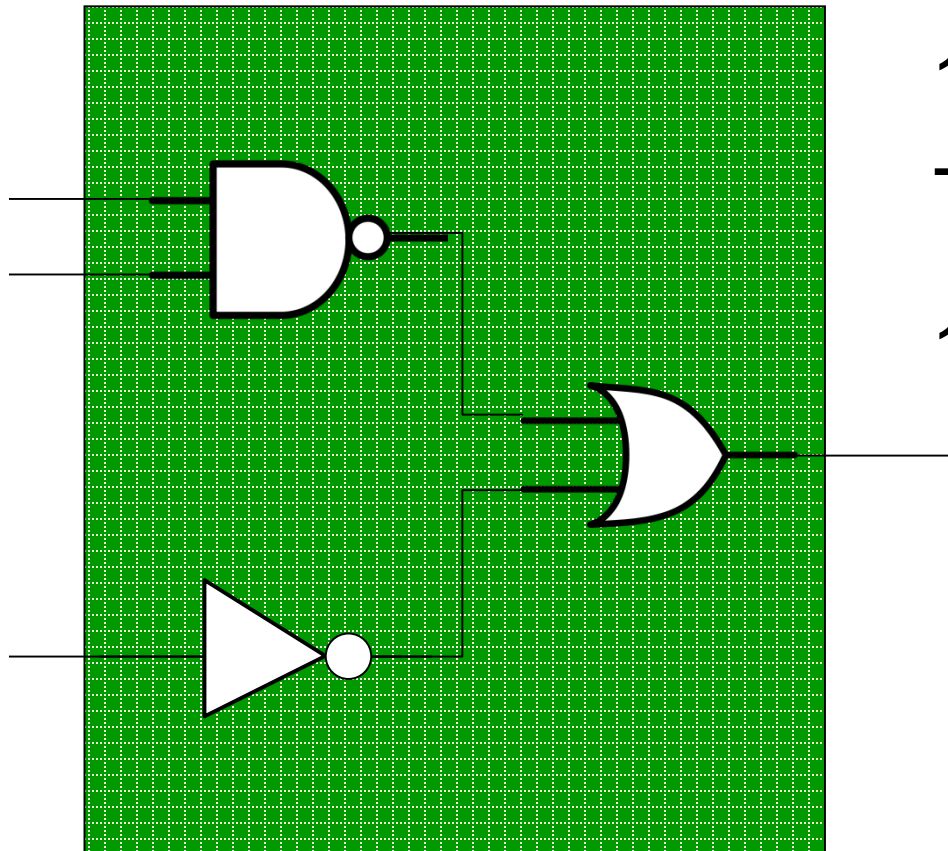
DIMENSIONNEMENT DE PORTES LOGIQUES

NAND à 4 entrées



2 NAND_2 + 1 OR_2
14 transistors

NAND à 3 entrées



1 NAND_2 + 1 OR_2
+ 1 NOT_1

12 transistors

NAND à 8 entrées



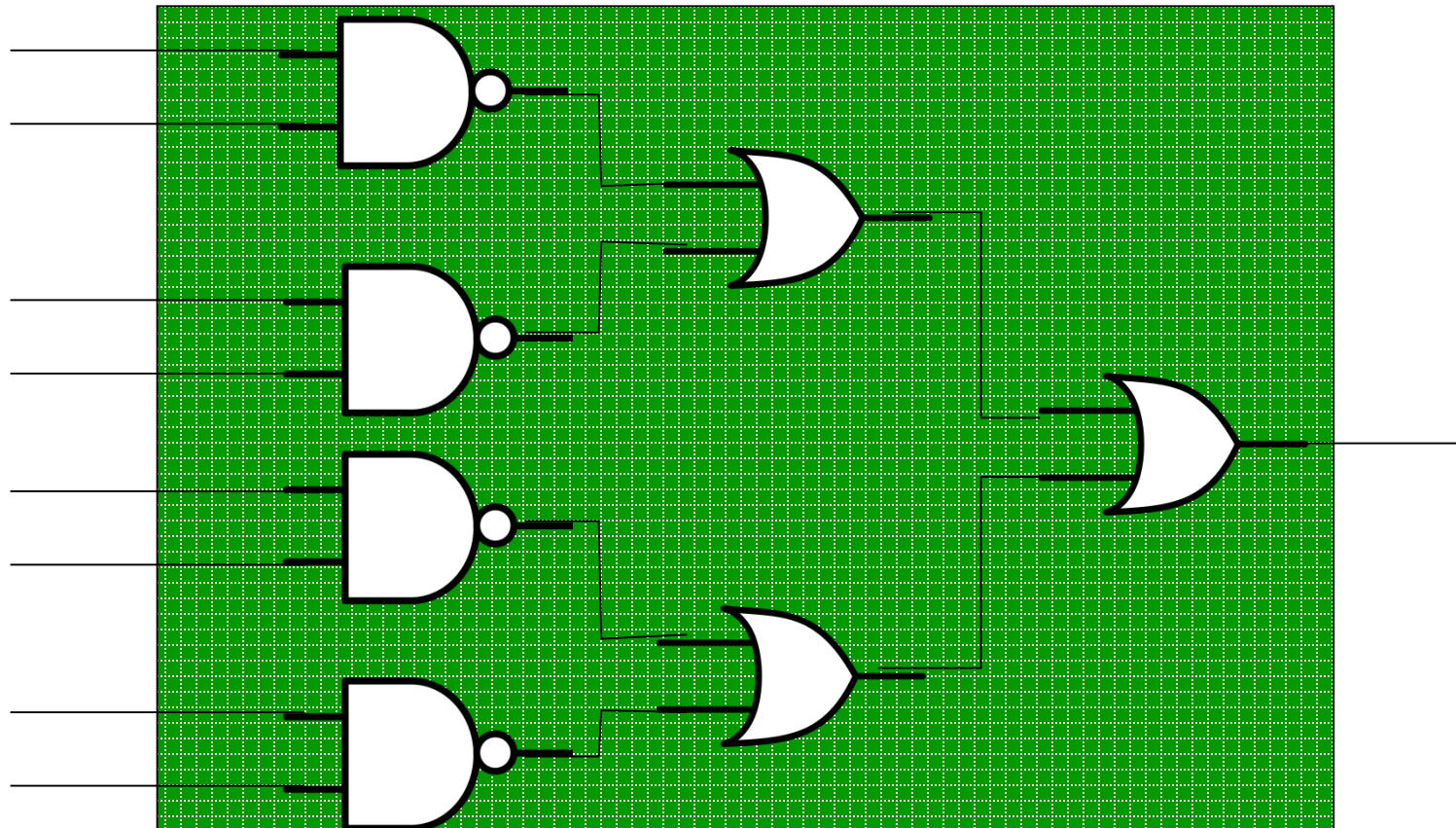
Combiens de portes ?

The diagram shows a large green rectangle representing an 8-input NAND gate. On the left side, there are eight horizontal lines representing input connections. On the right side, there is a single horizontal line representing the output connection. The text 'Combiens de portes ?' is centered within the green rectangle.

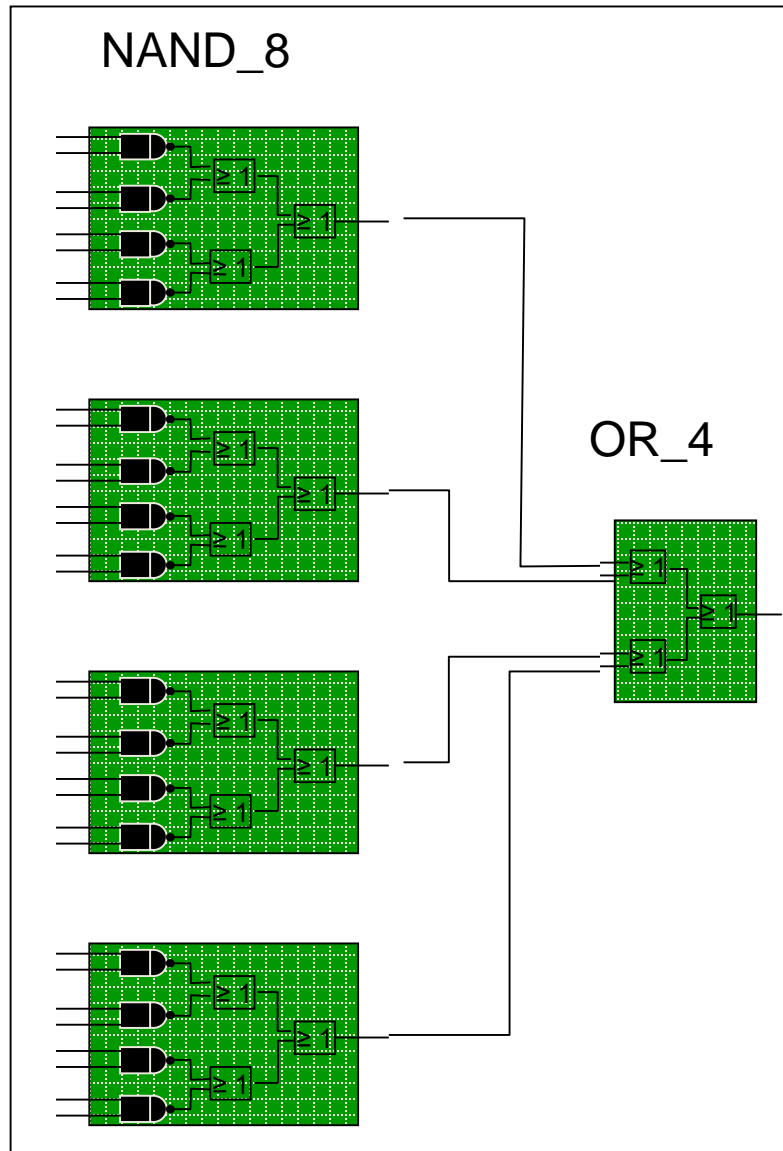
NAND à 8 entrées

4 NAND_2 + 3 OR_2

34 transistors



NAND à 32 entrées



NAND_32

16 NAND_2 + 15 OR_2
154 transistors

Une porte NAND à n entrées
demande $(n/2)$ portes NAND
et $(n/2) - 1$ portes OR à 2
entrées , et donc $5n-6$
transistors