

Circuits Logiques et Architectures de Machines

Informations pratiques

- Cours : Cécile Germain-Renaud
- Page Web : <http://www.lri.fr/~cecile/ENSEIGNEMENT/LAM/LAM.html>
- Contact : cecile.germain@lri.fr
- Contrôle des connaissances
 - Compte-rendus des TP : 0,1
 - Partiel : 0,3
 - Examen 0,6

2

Plan du cours

1. Représentation de l'information
2. Circuits combinatoires
3. Registres et automates
4. Mémoire
5. Architecture logicielle
6. Micro-Architecture
7. Introduction aux architectures avancées

3

1. L'information

Plan

- Définitions – Support
- Représentation des entiers
- Représentation des réels

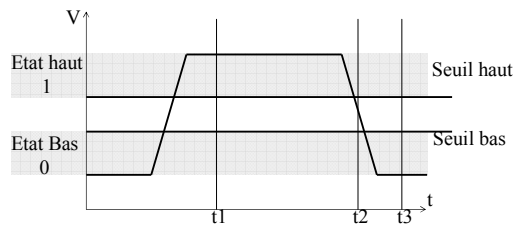
5

L'information

- Analogique : grandeur continue
- ou Numérique (= digitale) :
 - Information = connaissance d'un état parmi un nombre fini d'états possibles
 - Le cas le plus simple : 1 état parmi 2
- Support : grandeur physique continue - tension
- Le temps intervient pour obtenir une mesure pertinente de l'état

6

L'information



7

Codage - définitions

- 1 bit = quantité d'information liée à la connaissance d'1 état parmi 2 -> codage par 0 et 1
- Avec n bits, on peut coder 2^n états, donc la quantité d'information contenue dans la connaissance d'1 état parmi N est

$$\lceil \log_2(N) \rceil \text{ bits}$$

8

Codage - exemple

Etat	B2	B1	B0
France	0	0	0
GB	0	0	1
Allemagne	0	1	0
Espagne	0	1	1
Italie	1	0	0
Portugal	1	0	1
Grèce	1	1	0

9

Notations

- Binaire : 00101100
Peu pratique pour le traitement manuel
- Hexadécimale : 1 digit hexa par quartet
0x2C
- La notation hexadécimale ne préjuge pas du contexte d'interprétation
exemple : 'A' est codé par 0x41

10

Ordres de grandeurs

Ordre de grandeur	Puissance de 2	Puissance de 10
Kilo	10	3
Mega	20	6
Giga	30	9
Tera	40	12
Peta	50	15

11

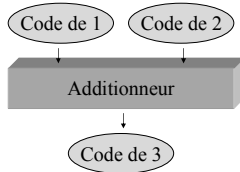
Quelle information ?

- Tous les processeurs
 - Nombres
 - Entiers naturels et relatifs
 - Réels
 - Caractères
 - Instructions
- Pourquoi pas d'autres ?

12

Quel codage?

Doit permettre de réaliser des opérateurs matériels rapides

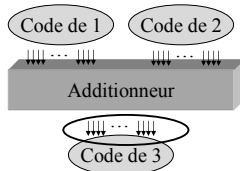


13

Codage et opérateurs

Sur un nombre fixe de bits

- caractéristique du processeur et de la technologie des CI
- typiquement 32 ou 64 bits pour le scalaire



14

Quelle information ?

- Certains processeurs représentent et traitent les vecteurs d'entiers et de réels
 - Processeurs « multimédia »
 - Console de jeux
- En général, pas les enregistrements (record), listes et autres types complexes

15

Codage des caractères

- **ASCII** American Standard Code for Information Interchange : 7 bits + 1 bit de parité
Caractères alphanumériques latins non accentués + caractères spéciaux
- **Latin -1** : 8 bits, standard ISO 8859-1. Caractères alphanumériques latins accentués
- **Unicode** : 16 bits, caractères d'alphabets variés

16

Plan

- Définitions - Support
- Représentation des entiers
- Représentation des réels

17

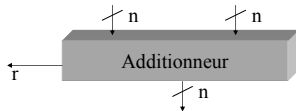
Codage des entiers naturels

- Sur n bits, écriture en base 2
Codage : division par les puissances de 2
Décodage : formule de définition
- Sur n bits, les nombres représentables sont
 $[0.. 2^n - 1]$
Sur 8 bits : $[0, 255]$

18

L'additionneur

- Additionneur : circuit matériel qui effectue l'addition naïve des entiers naturels sur n bits et tronque éventuellement la retenue
- Symbole : #
- Nom de l'opération : additionnage



19

L'additionneur

Le résultat du calcul dans l'additionneur peut être faux : retenue

Sur 3 bits

001 # 001 = 010 Correct : $1 + 1 = 2$
001 # 111 = 000 Faux : $1 + 7 \neq 0$

L'additionnage est tout simplement l'addition modulo 2^n , qui n'est définie que sur $\mathbb{Z} / 2^n\mathbb{Z}$.

20

Addition des naturels

Le résultat du calcul dans l'additionneur peut être faux

Un programme de test simple :

```
entier_non_signé : val ;  
val=1;  
tant que val <> 0 faire  
    val = val + 1;  
fin tant que  
afficher (" Quelle erreur !");
```

Le résultat dépend
- du langage de programmation
- du compilateur et des options

21

Multiplication et division par 2

- Multiplication par 2^k : décalage à gauche de k positions

$52 = 32 + 16 + 4$ 00110100
Décalage à gauche de 2 pos : 11010000
 $11010000 = 128 + 64 + 16 = 208 = 4 \times 52$

- Division entière par 2^k : décalage à droite de k positions
- Opérateur matériel très simple

22

Codage des entiers relatifs

Codage en complément à 2 sur n bits

- Les nombres représentés sont

$$[-2^{n-1} \dots 2^{n-1} - 1]$$

- Si $N \geq 0$, N est codé comme un entier naturel
- Si $N < 0$, N est codé par la représentation en naturel de

$$2^n + N$$

23

Codage en complément à 2 sur 3 bits

Nombre	Code
0	000
1	001
2	010
3	011
-1	De $8 - 1 = 7$, soit 111
-2	De $8 - 2 = 6$, soit 110
-3	De $8 - 3 = 5$, soit 101
-4	De $8 - 4 = 4$, soit 100

24

Addition des relatifs

Le résultat du calcul dans l'additionneur peut être faux : *overflow*

Sur 3 bits :

001 # 001 = 010 correct : $1 + 1 \rightarrow 2$
 001 # 110 = 111 correct : $1 + (-2) = -1$
 111 # 111 = 110 correct : $-1 + -1 = -2$
 011 # 001 = 100 faux : $3 + 1$ n'est pas codable
 110 # 101 = 011 faux : $-2 + (-3)$ n'est pas codable

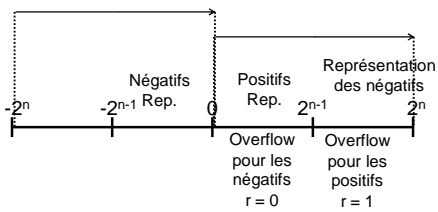
28

Addition des relatifs

1. L'addition des codes de deux relatifs N et M est égal à $N + M$ si et seulement si $N + M$ est codable
2. L'addition des codes de deux relatifs N et M est égal à $N + M$ si et seulement si
 - N et M sont de signes contraires
 - Ou bien N et M sont de même signe, et le bit de signe du résultat est égal à la retenue

29

Addition des relatifs



30

Codage des entiers naturels et relatifs : conclusion

- Avec les codages choisis, l'additionneur peut effectuer l'addition des naturels et des relatifs
- On verra au ch. 2 qu'il suffit de l'étendre modérément pour effectuer la soustraction
- Mais le résultat peut être faux s'il n'appartient pas à l'ensemble d'entiers représentables
 - Analyser l'application et le programme pour garantir que le résultat reste codable
 - Si ce n'est pas possible, spécifier le comportement sur overflow et utiliser les ressources de l'environnement pour le détecter et réaliser l'action spécifiée.

31

Plan

- Définitions – Support
- Représentation des entiers
- Représentation des réels

32

Codage des réels

- Problèmes :
 - Arrondir
 - Représenter équitablement des nombres très grands et très petits
- Principe : représentation normalisée par mantisse et exposant, avec un choix judicieux des valeurs de l'exposant

33

Mantisse et exposant

$$x = \pm m \cdot B^e$$

- m est la *mantisse*, e l'*exposant*, B la *base*
 m est un nombre deux-cimal
- Si B est connu, x peut être codé par (m, e)
- Exemple en notation décimale
 $2,76 = 2,76 \cdot 10^0 = 276 \cdot 10^{-2} = 27,6 \cdot 10^{-1} = \dots$
- Représentation normalisée (« scientifique ») :
1 seul chiffre, différent de 0, avant la virgule

34

Mantisse-exposant en base 2

- Le codage des réels utilise la représentation mantisse-exposant, avec $B = 2$
- Nombres deucimaux : de la forme $N/2^p$
 $1,10011_2 = 1 + 2^{-1} + 2^{-4} + 2^{-5} = 110011_2 \cdot 2^{-5}$
 $= 51 \cdot 2^{-5}$
Dans la suite, on ne note plus l'indice 2
- Normalisation : le chiffre avant la virgule est forcément un 1, qui sera implicite dans le codage de la mantisse pour économiser un bit

35

Codage IEEE 754 simple précision



E = interprétation en naturel
Pour $E \neq 0$ et 255, la valeur codée est

$$(-1)^s \cdot 2^{E-127} \cdot 1, f$$

Soit $e = E - 127$. Alors $-126 \leq e \leq 127$

36

Codage IEEE 754 simple précision



Que représente 0xC8900000 ?

1100 1000 1001 0000 ...0

$s = 1$; $E = 10010001 = 145$ donc $e = 18$

$f = 001 0000 \dots 0$ donc $m = 1 + 2^{-3}$

Le nombre codé est $-2^{18}(1 + 2^{-3})$

37

Codage IEEE 754 simple précision

- Codage de 2,5

$2,5 = 5 \times 2^{-1} = 101_2 \times 2^{-1} = 1,01_2 \times 2^1$

$E = 127 + 1 = 128$ $f = 01$

codage : 0 10000000 010...0 = 0x40200000

- Codage de 0,75

$0,75 = 3 \times 2^{-2} = (2+1) \times 2^{-2} = (1 + 2^{-1}) 2^{-1} = 1,1_2 \times 2^{-1}$

$E = 127 - 1 = 126$ $f = 1$

codage : 0 01111110 100...0 = 0x3F400000

38

L'addition des flottants

$2,5 = 2^1 \times 1,01_2$
 $0,75 = 2^{-1} \times 1,1_2$

1. Comparaison des exposants Différence 2

2. Décalage à droite de la mantisse du plus petit nombre $0,75 = 2^1 \times 0,011_2$

3. Somme des mantisses $1,01_2 + 0,011_2 = 1,101_2$

4. Normalisation du résultat Normalisé

5. Traitement cas exceptionnels overflow, underflow, zero Non

Résultat = $2^1 \times 1,101_2 = 3,25$

39

L'addition des flottants

$$\begin{aligned} 2,5 &= 2^1 \times 1,01_2 \\ 3,25 &= 2^1 \times 1,101_2 \end{aligned}$$

1. Comparaison des exposants Différence 0
2. Décalage à droite de la mantisse du plus petit nombre Pas de décalage
3. Somme des mantisses $1,101_2 + 1,01_2 = 10,111_2$
4. Normalisation du résultat $10,111_2 \times 2^1 = 1,0111_2 \times 2^2$
5. Traitement cas exceptionnels overflow, underflow, zero Non

$$\text{Résultat} = 2^2 \times 1,0111_2 = 5,75$$

40

L'addition des flottants

$$\begin{aligned} 2^{30} &= 2^{30} \times 1,01_2 \\ 3,25 &= 2^1 \times 1,101_2 \end{aligned}$$

1. Comparaison des exposants Différence 29
2. Décalage à droite de la mantisse du plus petit nombre Mantisse à 0
3. Somme des mantisses $1,101_2 + 0,0_2 = 1,101_2$
4. Normalisation du résultat Normalisé
5. Traitement cas exceptionnels overflow, underflow, zero Arrondi

$$\text{Résultat} = 2^{30}$$

41

Erreurs d'arrondi

- Le résultat d'une addition est souvent faux
- Mais le standard impose que le résultat soit le résultat exact arrondi, avec des choix possibles
 - Au plus près / Valeur Inférieure / Valeur supérieure
 - Mais $2^{30} + 3,25$ a pour résultat 2^{30}
- Réalisé avec 3 bits supplémentaires seulement
- En langage de haut niveau, options de compilation

42

Erreurs d'arrondi

- L'addition n'est plus associative

$$(-2^{30} + 2^{30}) + 3,25 = 3,25$$

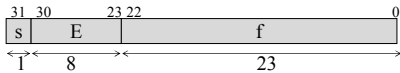
$$-2^{30} + (2^{30} + 3,25) = 0$$

- Le traitement des erreurs d'arrondi dans les méthodes numériques est une composante essentielle de l'informatique numérique
 - Très bien connu pour l'algèbre linéaire et encapsulé dans des bibliothèques dont l'utilisation est impérative
 - Beaucoup moins bien en simulation

43

Codage IEEE 754 simple précision

Les cas exceptionnels



Nom	E	f	valeur
Dénormalisé	0	Non 0	$(-1)^s \cdot 2^{-127} \cdot 0, f$
Zéro	0	0	0
Infini	255	Non 0	$(-1)^s \cdot \infty$
NaN	255	0	NaN

44

La norme IEEE 754

- Codage simple précision (32 bits), double précision (64 bits), simple étendue et double étendue
- Spécification des arrondis suivant les opérations, de l'arithmétique étendue ; Objectif : reproductibilité
- Traitement des cas exceptionnels

David Goldberg. What every computer scientist should know about floating point arithmetic

45

Conclusion

- L'interprétation d'une chaîne de bits dépend du contexte.
0xC8900000 -> $-2^{18}(1 + 2^{-3})$ en flottant
 -> $12 \times 16^7 + 8 \times 16^6 + 9 \times 16^5$ en naturel sur
 32 bits
 -> ... en relatif
- Comment le matériel reconnaît-il le contexte : voir deuxième partie
- La validité des calculs doit être démontrée a priori

46

2. Circuits combinatoires

Plan

- Algèbre de Boole et portes logiques
- Représentations des fonctions booléennes
 - Formes normales
 - Minimisation
- Réalisations
 - Multiplexeurs et décodeurs
 - PLA
 - L'additionneur

48

Introduction

- Information Etat Haut - Etat Bas
- Abstraction :
 - Etat Haut -> 1
 - Etat Bas -> 0

Propriétés formelles de la représentation abstraite en vue d'une réalisation matérielle

49

Algèbre de Boole

Soit E un ensemble non vide, muni de deux opérations binaires $+$ et \cdot , d'une opération unaire notée $\bar{}$, et deux éléments particuliers de E notés 0 et 1. $(E, 0, 1, +, \cdot, \bar{})$ est une algèbre de Boole si

- (i) les opérations $+$ et \cdot sont commutatives et associatives
- (ii) 0 est neutre pour $+$ et 1 pour \cdot
- (iii) $+$ et \cdot sont distributives l'une sur l'autre
- (iv) $\forall a \in E, a \cdot \bar{a} = 0$ et $a + \bar{a} = 1$

50

L'algèbre de Boole des circuits logiques

- $E = \{0, 1\}$ et $+$, \cdot , $\bar{}$ sont définies formellement par :

+	0	1
0	0	1
1	1	1

\cdot	0	1
0	0	0
1	0	1

$\bar{}$	
0	1
1	0

- Interprétation logique, au sens philosophique,
 \cdot = ET, $+$ = OU, $\bar{}$ = NEGATION (NOT)
- Notation : priorité de $\bar{}$ sur $+$

51

Propriétés

- Si $(E, 0, 1, +, \cdot, \bar{})$ est une algèbre de Boole, $(E, 1, 0, \cdot, +, \bar{})$ est aussi une algèbre de Boole.
- $\overline{\overline{a}} = a$
- $a + a = a$ $a \cdot a = a$
- $a + 1 = 1$ $a \cdot 0 = 0$
- $a + 0 = a$ $a \cdot 1 = a$
- Règles de De Morgan $\forall a, b \in E, \overline{a + b} = \bar{a} \cdot \bar{b}$
 $\forall a, b \in E, \overline{a \cdot b} = \bar{a} + \bar{b}$

52

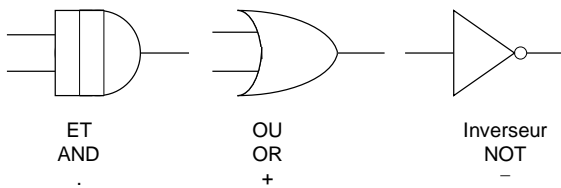
D'autres algèbres de Boole

- $P(E, \emptyset, E, \cup, \cap)$
- Les n-uplets de booléens
- Dans la suite, on considère l'algèbre à 2 éléments $(B, 0, 1, +, \cdot, \bar{})$.

53

Portes logiques

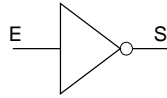
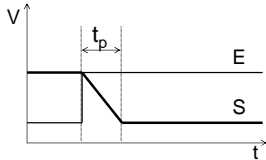
Porte logique : circuit qui réalise une fonction élémentaire



54

Portes logiques et algèbre de Boole

- Les portes logiques ne réalisent les fonctions logiques qu'à certains points dans le temps
- Le *temps de propagation* d'un circuit est le temps qui sépare le positionnement des entrées de celui de la sortie
- Ordres de grandeur
 - Lumière dans le vide : 4ns/m – nano seconde – 10^{-9}
 - Temps de commutation: pico seconde – ps – 10^{-12}



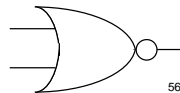
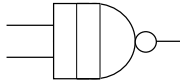
55

Portes NAND et NOR

- Les portes ET et OU ne sont pas les plus simples à réaliser en technologie CMOS
- Les vraies portes élémentaires sont les portes NAND, NOR et NOT

$$\text{NAND}(a, b) = \overline{a \cdot b}$$

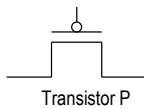
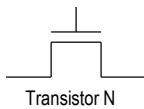
$$\text{NOR}(a, b) = \overline{a + b}$$



56

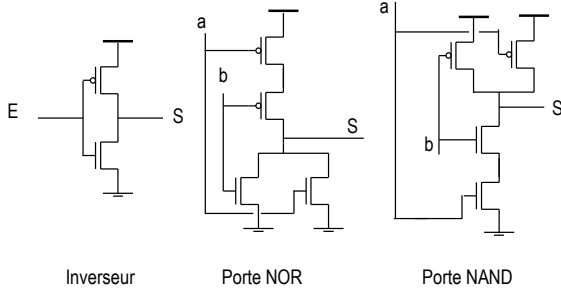
Technologie CMOS

- CMOS = Complementary Metal-oxide semiconductor
- Ici, transistor = interrupteur
- Le transistor P est ouvert si G=haut, fermé si G = bas
- Le transistor N est ouvert si G= bas, fermé si G = haut



57

Portes



Inverseur

Porte NOR

Porte NAND

58

Portes NAND et NOR

- NAND et NOR sont fonctionnellement complets : on peut réaliser ET, OU et NOT a partir de chacun d'entre eux

$$\bar{a} = \bar{a.1} = \text{NAND}(a, 1)$$

$$\bar{a} = \bar{a.a} = \text{NAND}(a, a)$$

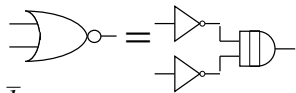
$$a.b = \overline{\bar{a}.\bar{b}} = \text{NOT}(\text{NAND}(a, b))$$

$$a + b = \overline{\bar{a}.\bar{b}} = \text{NAND}(\text{NOT}(a), \text{NOT}(b))$$

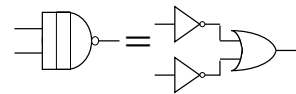
59

Portes NAND et NOR : la règle de De Morgan

$$\forall a, b \in E, \overline{a+b} = \bar{a}.\bar{b}$$



$$\forall a, b \in E, \overline{a.b} = \bar{a} + \bar{b}$$

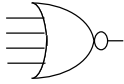


60

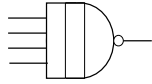
NAND_k et NOR_k

$$\text{NAND}_k(x_1, \dots, x_k) = \overline{x_1 \cdot x_2 \cdot \dots \cdot x_k}$$

$$\text{NOR}_k(x_1, \dots, x_k) = \overline{x_1 + x_2 + \dots + x_k}$$



NOR4



NAND4

Mais les contraintes technologiques limitent à un petit nombre d'entrées.

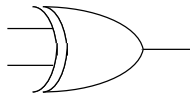
61

Le OU exclusif - XOR

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

$$a \oplus b = \bar{a}.b + a.\bar{b}$$

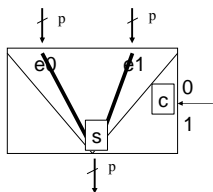
Porte XOR



62

Le multiplexeur

- Définition : if c then e0 else e1
- Expression logique : $s = \bar{c}.e0 + c.e1$
- Mais réalisé directement



63

Plan

- Algèbre de Boole et portes logiques
- Représentations des fonctions booléennes
 - Formes normales
 - Minimisation
- Réalisations
 - Multiplexeurs et décodeurs
 - PLA
 - L'additionneur

64

Fonctions booléennes

- Fonctions de $\{0,1\}^n$ vers $\{0,1\}$
- Représentations
 - En extension : tables de vérité
 - Algébrique : $f(a,b,c) = b + a.c$
 - La représentation algébrique n'est pas unique :
 $a + a = a$

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

65

Exemples

- Bit de parité
 $c = !a + b + a.!b = \text{XOR}(a,b)$
- Convertisseurs naturel-Gray
- Une fonction g de 4 variables qui est à 1 ssi le nombre de 1 est égal au nombre de 0

...

66

Contexte : expressions booléennes

- Toute expression de n variables booléennes
- Grâce aux lois de De Morgan, on peut considérer que les négations ne concernent que les variables individuelles
- Forme disjonctive normale (FDN)
OU de ET
 $a \cdot b + a \cdot c \cdot d$
- Forme conjonctive normale (FCN)
ET de OU
 $(a + b) \cdot (a + c + d)$

67

Contexte : expressions booléennes

- FCN : chaque clause exprime une contrainte
- Question fondamentale : satisfiabilité
- Satisfiabilité d'une expression en FDN
au moins un ET ne contient pas une variable et son complément
- Le problème général de la satisfiabilité est NP complet

68

Circuits combinatoires

- Dispositif matériel qui calcule = réalise une ou des fonctions booléennes
 $f_0(e_0, e_1, \dots, e_{n-1}), \dots, f_{p-1}(e_0, e_1, \dots, e_{n-1})$
- Synthèse logique : méthodes de conception des circuits combinatoires
- Compromis temps de propagation / encombrement / facilité de conception

69

Représentation des fonctions booléennes

- Deux représentations canoniques
 - Forme conjonctive normale et forme disjonctive normale
 - Une fonction booléenne a une et une seule forme conjonctive normale, et une et une seule forme disjonctive normale
 - Technologiquement utilisables
- Formes minimisées suivant des critères technologiques

70

Théorème de Shannon

$$f(x_1, \dots, x_n) = \bar{x}_1 \cdot f(0, x_2, \dots, x_n) + x_1 \cdot f(1, x_2, \dots, x_n)$$

Les deux membres sont identiques pour $x_1 = 0$ et $x_1 = 1$

- Applications
 - Ecrire la fonction comme un polynôme aux x_i et x_i complémentés (FDN, FCN)
 - Exprimer l'expression à partir de fonctions multiplexeurs

71

Forme disjonctive normale

Un terme produit (minterm) est un produit de toutes les variables d'entrées, complémentées ou non.

- Pour n variable d'entrée, il y a 2^n minterms
- Chaque minterm correspond à une ligne de la table de vérité. Chaque variable est complémentée si dans cette ligne sa valeur est 0, non complémentée sinon

$$\begin{aligned} m_0 &= \bar{x}_1 \cdot \bar{x}_0 \\ m_1 &= \bar{x}_1 \cdot x_0 \\ m_2 &= x_1 \cdot \bar{x}_0 \\ m_3 &= x_1 \cdot x_0 \end{aligned}$$

x0	x1	m3	m2	m1	m0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

72

Forme disjonctive normale

La forme disjonctive normale d'une fonction est le OU des minterms pour lesquels la fonction vaut 1

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

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

73

Forme disjonctive normale

La forme disjonctive normale d'une fonction est le OU des minterms pour lesquels la fonction vaut 1

Notation : $\sum m_i$

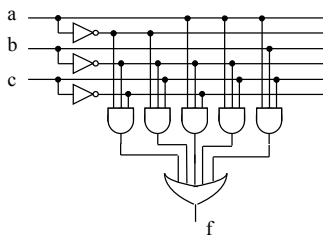
Exemple

$$g = m_3 + m_5 + m_6 + m_9 + m_{10} + m_{12}$$

74

Forme disjonctive normale et portes logiques

Logique à 2 niveaux : OU (ET)



75

Forme disjonctive normale et portes logiques

La forme disjonctive normale peut aussi s'exprimer par un NAND de NAND, en utilisant éventuellement des portes $NAND_k$

$$\begin{aligned}
 f &= \overline{a} \overline{b} \overline{c} + \overline{a} \overline{b} c + \overline{a} b \overline{c} + \overline{a} b c + a \overline{b} \overline{c} + a \overline{b} c + a b \overline{c} + a b c \\
 &= \overline{\overline{a} \overline{b} \overline{c} + \overline{a} \overline{b} c + \overline{a} b \overline{c} + \overline{a} b c + a \overline{b} \overline{c} + a \overline{b} c + a b \overline{c} + a b c} \\
 &= NAND_4(NAND_3(\overline{a}, \overline{b}, \overline{c}), NAND_3(\overline{a}, \overline{b}, c), \\
 &\quad NAND_3(\overline{a}, b, \overline{c}), NAND_3(\overline{a}, b, c), \\
 &\quad NAND_3(a, \overline{b}, \overline{c}), NAND_3(a, \overline{b}, c), NAND_3(a, b, \overline{c}), NAND_3(a, b, c))
 \end{aligned}$$

76

Forme conjonctive normale

Un terme somme (maxterm) est une somme de toutes les variables d'entrées, complémentées ou non.

- Chaque maxterm correspond à une ligne de la table de vérité. Chaque variable est complémentée si dans cette ligne sa valeur est 1, non complémentée sinon
- La forme conjonctive normale d'une fonction est le ET des maxterms pour lesquels la fonction vaut 0.
- La forme conjonctive normale peut s'écrire comme un NOR de NOR

77

Forme conjonctive normale

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

$$f = (a + \overline{b} + c) \cdot (a + \overline{b} + \overline{c}) \cdot (\overline{a} + \overline{b} + c)$$

78

Minimisation

- Contrainte : toujours logique à 2 niveaux
- Moins de minterms : plus petit OU
- Moins de variables dans chaque minterm : plus petits ET

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

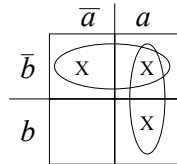
$$\bar{a}\bar{b} + a\bar{b} + ab = a + \bar{b}$$

79

Principe des tables de Karnaugh

Exploiter la propriété $a + \bar{a} = 1$ par une représentation graphique

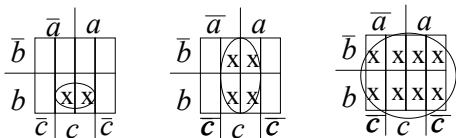
$$\bar{a}\bar{b} + a\bar{b} + ab = (\bar{a}\bar{b} + a\bar{b}) + (a\bar{b} + ab) = \bar{b} + a$$



80

Tables de Karnaugh : exemples de regroupements

- Trois variables



bc

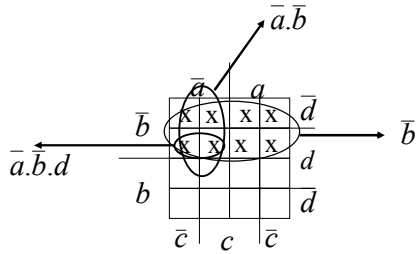
c

1

81

Tables de Karnaugh : exemples

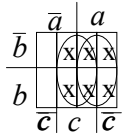
- Quatre variables



82

Tables de Karnaugh : exemples

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



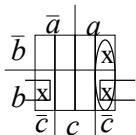
$$f = a + c$$

La fonction peut s'écrire comme le OU des regroupements par puissances de 2

83

Tables de Karnaugh

- Attention : grille torique

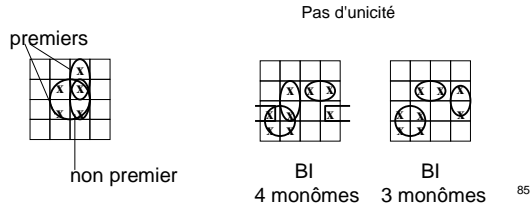


$$f = a.\bar{c} + b.c$$

84

Minimisation : définitions

- Monôme premier : qui n'est contenu dans aucun autre
- Base *irrédundante* : ensemble de monômes premiers
 - (a) Dont la somme est égale à la fonction
 - (b) Telle que si on enlève un monôme, (a) n'est plus vraie



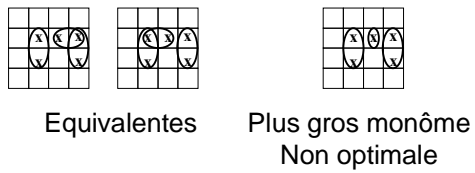
Minimisation : définitions

- Base irrédundante minimale : en nombre de monômes, puis en taille de monômes
- Algorithme :
 - Générer tous les monômes premiers
 - Générer toutes les bases irrédundantes par essais successifs
 - Choisir une base minimale (pas unique)
- Le problème est NP-complet

86

Bases irrédundantes minimales

Pas d'unicité



87

Minimisation manuelle

- Effectuer les regroupements par ordre décroissant de taille
- Choisir les regroupements qui éliminent le plus grand nombre de x sans redondance
- Tous les x doivent être pris en compte

88

Afficheur 7-segments



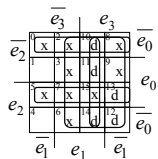
89

Fonctions incomplètement spécifiées

La valeur de la fonction n'est pas significative pour certaines valeurs des variables d'entrée.



Le segment a est allumé pour 0, 2, 3, 5, 6, 7, 8, 9



$$a = e_1 + e_3 + \overline{e_2} \cdot \overline{e_0} + e_2 \cdot e_0$$

90

Plan

- Algèbre de Boole et portes logiques
- Représentations des fonctions booléennes
 - Formes normales
 - Minimisation
- Réalisations
 - Multiplexeurs et décodeurs
 - PLA
 - L'additionneur

91

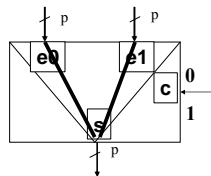
Les fonctions élémentaires

- Les opérateurs logiques : par les portes ET, OU, NAND, NOR, XOR...
- Portes complexes, dépendant de la technologie
- La sélection : par le multiplexeur ou le décodeur

92

Les Multiplexeurs

- Fonction : sélection
- Le + simple : 1 parmi 2
 - si $c == 0$
 - alors
 - $s = e0$
 - sinon
 - $s = e1$

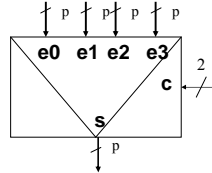


93

Les Multiplexeurs

- 1 parmi 2^n
- $2^n \times p$ entrées de données,
- n entrées de contrôle,
- $1 \times p$ sortie

$$S = e_i \text{ si } c_{n-1} \dots c_0 = i$$

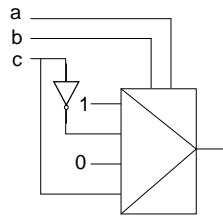


94

Multiplexeurs

- Toute fonction Booléenne de n variables peut être implémentée avec un multiplexeur 1 parmi 2^{n-1} .

a	b	c	f	
0	0	0	1	$S = 1$
0	0	1	1	
0	1	0	1	$S = \bar{c}$
0	1	1	0	
1	0	0	0	$S = 0$
1	0	1	0	
1	1	0	0	$S = c$
1	1	1	1	



95

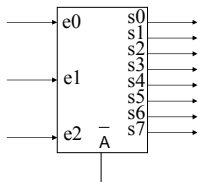
Multiplexeurs

- Pour une fonction de n variables, par exemple $f(A,B,C,D)$
 - Avec un MUX 1 parmi 2^{n-1} donc $n-1$ lignes de sélection
 - Définir un ordre des variables, par exemple $e3 = A$, $e2 = B$, $e1 = C$, $e0 = D$
 - Les $n-1$ bits de poids fort vont vers les $n-1$ lignes de sélection, par exemple A,B,C
 - Pour deux lignes consécutives, les valeurs possibles de la sortie sont soit 0, soit 1, soit !e0, soit !e0.
 - Connecter 0, e0, !e0, ou 1 à chaque entrée d'après le résultat de l'étape précédente.

96

Les décodeurs

- n entrées de données, 2^n sorties = les minterms Donc, 1 seule sortie active un instant donné
 $S_i = 1$ si $e_{n-1} \dots e_0 = i$
- Toute fonction booléenne peut être générée avec un décodeur et un OR : FDN



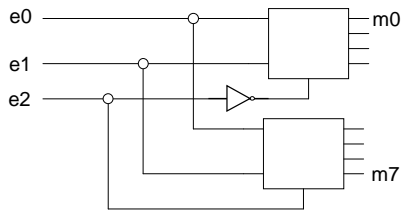
Exemple d'utilisation :
Fonction g

$$S_1 = \bar{e}_2 \cdot \bar{e}_1 \cdot e_0$$

97

Réalisation structurée

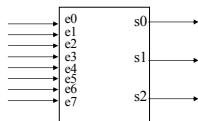
- Un décodeur 3:8 à partir de décodeurs 2:4



98

Les encodeurs

- Fonction réciproque du décodeur
 Une seule entrée à 1, écriture binaire du numéro de l'entrée



99

Les encodeurs

- Fonction réciproque du décodeur
Une seule entrée à 1, écriture binaire du numéro de l'entrée active
- Exemple : encodeur 4:2

e3	e2	e1	e0	s1	s0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

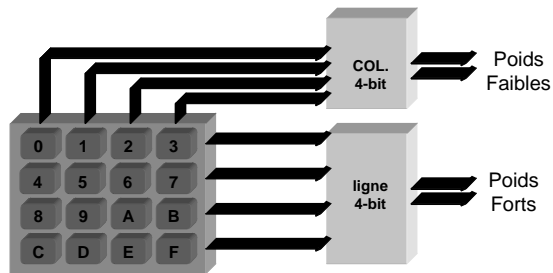
Réalisation
minimisée

$$s0 = e1 + e3$$

$$s1 = e2 + e3$$

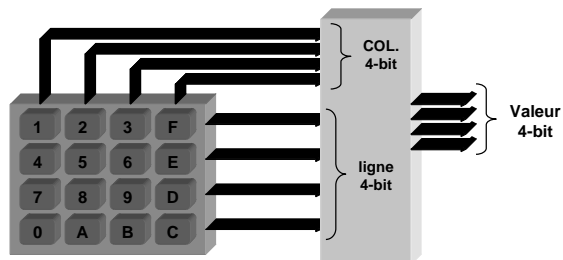
100

Organisation matricielles



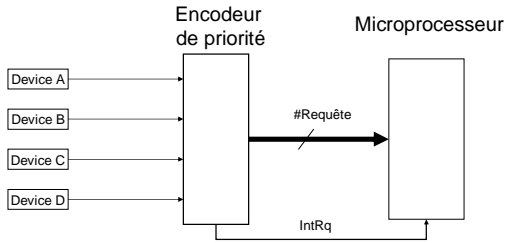
101

Clavier hexadécimal



102

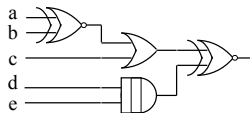
Gestion d'interruption



106

Fonctions booléennes quelconques

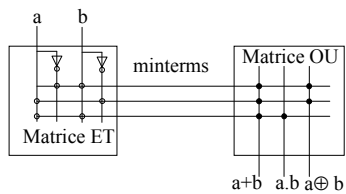
- Une fonction de quelques variables, pas de régularité : logique "anarchique"
 $(a \oplus b + c) \oplus (d.e)$



107

Fonctions booléennes quelconques

- Plusieurs fonctions de plusieurs variables, pas de régularité : Programmable Logic Array (PLA)

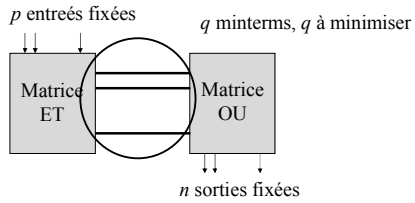


108

La structure PLA

- Optimise la surface et la faisabilité

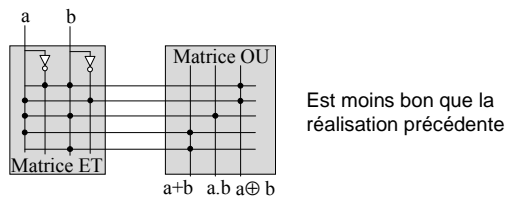
Existe comme circuit indépendant configurable par claquage des connexions ET et OU (PLA) ou seulement ET (PAL)



109

La structure PLA

- Minimisation globale sur l'ensemble des fonctions : la minimisation individuelle peut être contre-productive



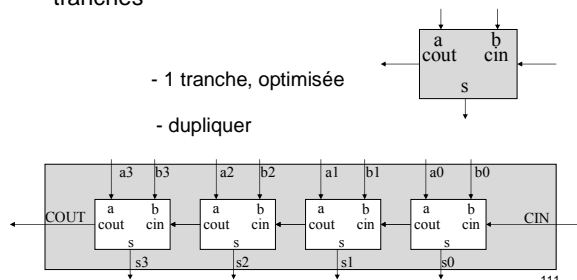
110

Logique en tranches

- $n \times 1$ fonction de plusieurs variables : logique en tranches

- 1 tranche, optimisée

- dupliquer



111

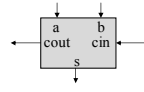
L'additionneur

- L'additionneur 1 bit

a	b	Cin	S	Cout
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

$$S = \bar{a}.\bar{b}.Cin + \bar{a}.b.\bar{Cin} + a.\bar{b}.\bar{Cin} + a.b.Cin$$

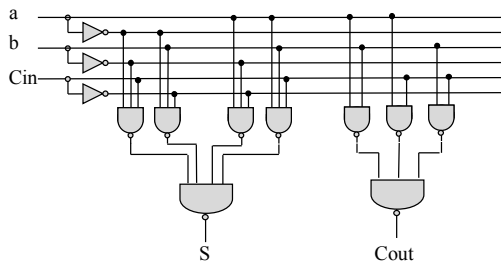
$$Cout = a.b + a.Cin + b.Cin$$



112

L'additionneur 1 bit en portes NAND_k

$$S = \bar{a}.\bar{b}.Cin + \bar{a}.b.\bar{Cin} + a.\bar{b}.\bar{Cin} + a.b.Cin \quad Cout = a.b + a.Cin + b.Cin$$



113

L'additionneur 1 bit

Hypothèse : $1t$ par porte NAND_k ou NOR_k

$$S = \bar{a}.\bar{b}.Cin + \bar{a}.b.\bar{Cin} + a.\bar{b}.\bar{Cin} + a.b.Cin$$

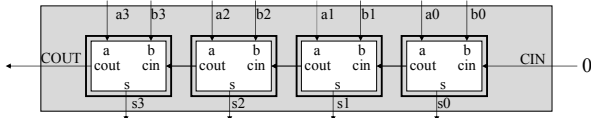
Temps de propagation : $3t$

$$Cout = a.b + a.Cin + b.Cin$$

Temps de propagation : $2t$

114

L'additionneur n bits à propagation de retenue



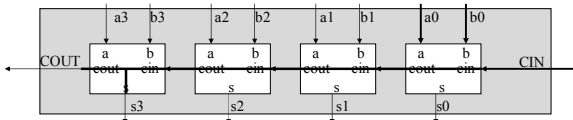
Problème : temps de propagation de la retenue

- $2nt$ pour COUT
- $(2n + 1)t$ pour S_n

115

Chemin critique

Plus long chemin (en temps) des entrées vers une sortie



116

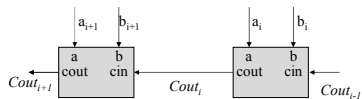
Retenue anticipée

- Accélérer le calcul de la retenue de sortie

Soient $P = a + b$ propager une retenue à 1

$G = a \cdot b$ générer une retenue

Alors $Cout = G + P \cdot Cin$



117

Retenue anticipée

Si $P = a + b$ et $G = a.b$
 alors $Cout = G + P . Cin$

En effet

$$Cout = a.b + a.Cin + b.Cin$$

$$= G + P . Cin$$

Remarque: vrai aussi pour $P = a \text{ XOR } b$

118

Circuit anticipateur de retenue

- Calcul en 2 couches logiques de la retenue sortante.

Exemple sur 4 bits

$$Cout_3 = G_3 + P_3 . Cout_2$$

$$= G_3 + P_3 . (G_2 + P_2 . Cout_1)$$

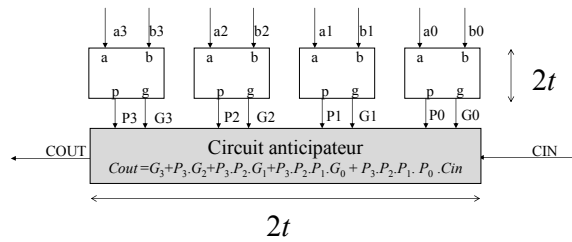
$$= G_3 + P_3 . (G_2 + P_2 . (G_1 + P_1 . Cout_0))$$

$$= G_3 + P_3 . (G_2 + P_2 . (G_1 + P_1 . (G_0 + P_0 . Cout_0)))$$

$$Cout_3 = G_3 + P_3 . G_2 + P_3 . P_2 . G_1 + P_3 . P_2 . P_1 . G_0 + P_3 . P_2 . P_1 . P_0 . Cout_0$$

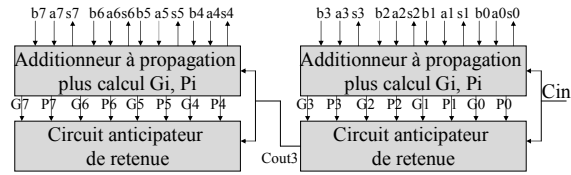
119

Circuit anticipateur de retenue



120

Additionneur à retenue anticipée – Version 1



S3 9t, le calcul anticipé ne change rien
 Cout3 2t pour G_i, P_i
 plus 2t pour le circuit anticipateur : 4t
 S7 temps de calcul de $Cout3$ plus additionneur à propagation : 13t
 Au lieu de 17t dans l'additionneur à propagation simple 121

Additionneur à retenue anticipée – Version 2

- Calcul anticipé de toutes les retenues

$$S_i = \overline{G_i} \cdot P_i \oplus Cout_{i-1}$$

$$Cout_0 = G_0 + P_0 \cdot Cout_{-1}$$

$$Cout_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot Cout_{-1}$$

$$Cout_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot Cout_{-1}$$

etc.

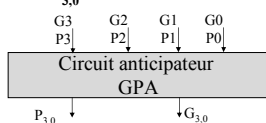
- Toutes les retenues se calculent en 4t
 2t pour les G_i, P_i
 2t pour chaque retenue indépendamment
- Tous les S_i en 7t : 4 pour les retenues et 3 pour le calcul de S_i

122

Additionneur à retenue anticipée

- Problème : l'hypothèse 2 couches logiques = 2t n'est valide que pour un petit nombre d'entrées, par exemple 4.
- Le même principe peut s'appliquer récursivement

$$Cout_3 = \overbrace{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0}^{G_{3,0}} + \overbrace{P_3 \cdot P_2 \cdot P_1 \cdot P_0}^{P_{3,0}} \cdot Cout_{-1}$$



123

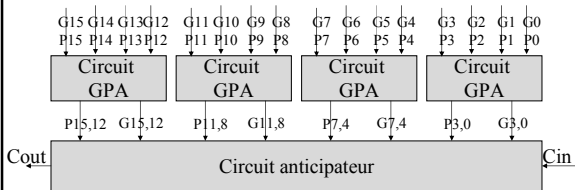
Retenue anticipée par blocs

$$\begin{aligned}
 \text{Cout}_3 &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot \text{Cout}_{-1} \\
 \text{Cout}_3 &= G_{3,0} + P_{3,0} \cdot \text{Cout}_{-1} \\
 \text{Cout}_7 &= G_7 + P_7 \cdot G_6 + P_7 \cdot P_6 \cdot G_5 + P_7 \cdot P_6 \cdot P_5 \cdot G_4 + P_7 \cdot P_6 \cdot P_5 \cdot P_4 \cdot \text{Cout}_3 \\
 \text{Cout}_7 &= G_{7,4} + P_{7,4} \cdot \text{Cout}_3 \\
 &= G_{7,4} + P_{7,4} \cdot (G_{3,0} + P_{3,0} \cdot \text{Cout}_{-1}) \\
 \dots \\
 \text{Cout}_{15} &= G_{15,12} + P_{15,12} \cdot G_{11,8} + P_{15,12} \cdot P_{11,8} \cdot G_{7,4} + \\
 &\quad P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0} + P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot \text{Cout}_{-1}
 \end{aligned}$$

124

Retenue anticipée par blocs

$$\begin{aligned}
 \text{Cout}_{15} &= G_{15,12} + P_{15,12} \cdot G_{11,8} + P_{15,12} \cdot P_{11,8} \cdot G_{7,4} + \\
 &\quad P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0} + P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot \text{Cout}_{-1}
 \end{aligned}$$

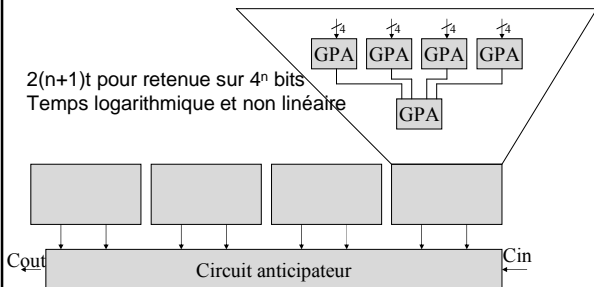


6t

125

Retenue anticipée par blocs

$2(n+1)t$ pour retenue sur 4^n bits
Temps logarithmique et non linéaire

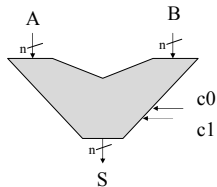


126

L'UAL

- Addition, soustraction, opérations logiques bit à bit
- Implique entrée de sélection : commande

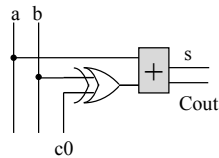
c1	c0	S
0	0	A AND B
0	1	A OR B
1	0	A + B
1	1	A - B



127

Une tranche 1 bit de l'UAL

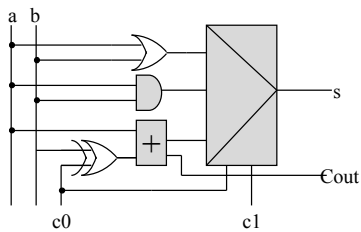
- Soustraction = ajouter l'opposé = complément bit à bit + 1
 - Choix : sur c0
 - Si c0 = 0, ajouter b
 - Si c0 = 1, ajouter Not(b)
- Mais
 $a \text{ XOR } 0 = a$
 $A \text{ XOR } 1 = !a$



128

Une tranche 1 bit de l'UAL

- Version naïve, multiplexeurs et propagation de retenue



129

3. Circuits séquentiels : registres et automates

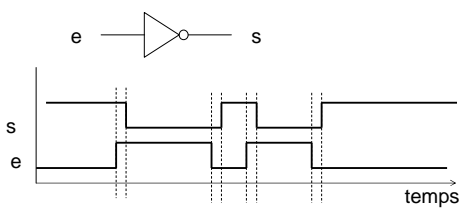
Plan

- Motivation
- Bascules et registres
 Conception des circuits de mémorisation
- Automates
 Utilisation des circuits de mémorisation

131

Comportement temporel des circuits combinatoires

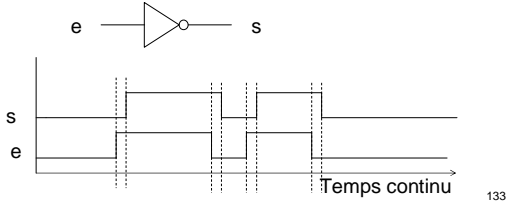
- Les entrées suivent les sorties - avec un temps de retard



132

Objectif des circuits séquentiels

- Mémorisation en circuiterie logique
- Définir les valeurs observables dans les circuits en discrétisant le temps

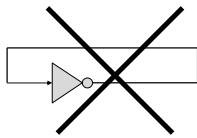


133

Conception

Bascule : unité de mémorisation 1 bit

- Principe : utiliser des circuits combinatoires rebouclés dans un état stable. Réservé à l'intérieur des registres.



134

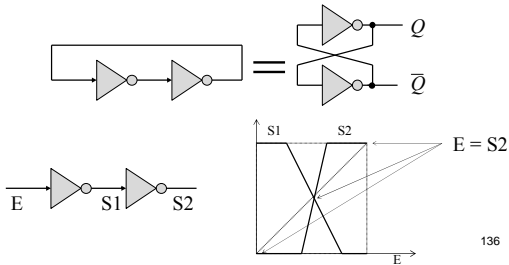
Plan

- Motivation
- Bascules et registres
 - Conception des circuits de mémorisation
- Automates
 - Utilisation des circuits de mémorisation

135

Le Bistable

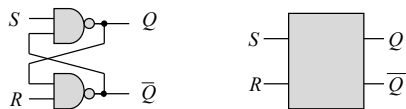
- 2 points fixes stables, 1 point instable. Pas d'entrées, irréversible.



136

La bascule RS

- Positionnement ou mémorisation

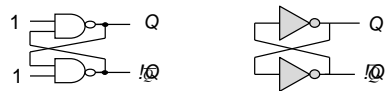


137

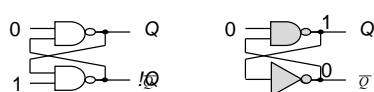
La bascule RS

Nand (x, 1) = !x Nand (x, 0) = 1

- R = S = 1 : Mémorisation de la valeur courante



- R = 1 S = 0 : Q = 1 et !Q = 0 Écriture d'un 1

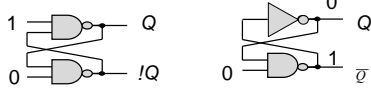


138

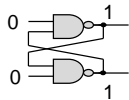
La bascule RS

$\text{Nand}(x, 1) = !x$ $\text{Nand}(x, 0) = 1$

- $R = 0$ $S = 1$: $Q = 0$ et $!Q = 1$ Écriture d'un 0



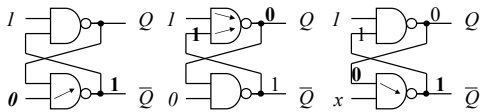
- $R = 0$ $S = 0$: configuration interdite, les deux sorties ne sont plus complémentaires 1, 1



139

La bascule RS

- Écriture d'un 0 : détail du fonctionnement

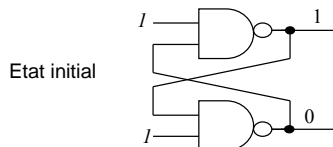


A la fin de la deuxième étape, Q à 0 verrouille le 1 sur $!Q$
 R peut changer ensuite : R doit être maintenu à 0 au moins $2t_p$
 De même, pour l'écriture d'un 1, S doit être maintenu à 0 au moins $2t_p$

140

La bascule RS

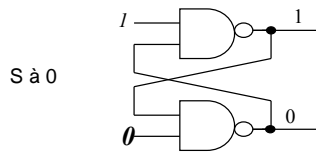
- Écriture d'un 0 : fonctionnement pathologique si non - respect des t_p



141

La bascule RS

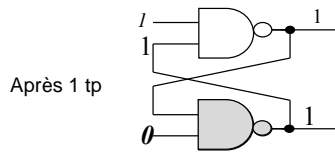
- Ecriture d'un 0 : fonctionnement pathologique si non - respect des tp



142

La bascule RS

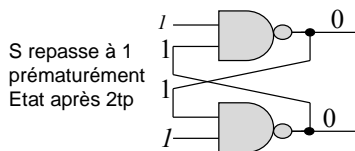
- Ecriture d'un 0 : fonctionnement pathologique si non - respect des tp



143

La bascule RS

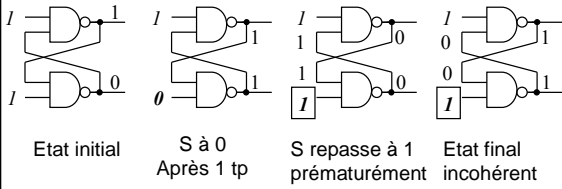
- Ecriture d'un 0 : fonctionnement pathologique si non - respect des tp



144

La bascule RS

- Ecriture d'un 0 : fonctionnement pathologique si non - respect des t_p



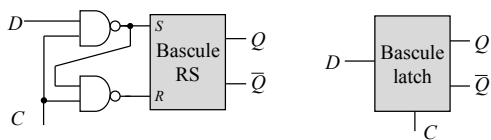
145

La bascule RS

- Ecriture d'un 1 : $S = 0, R = 1$
 - Ecriture d'un 0 : $S = 1, R = 0$
- Données et commande sont mélangées

146

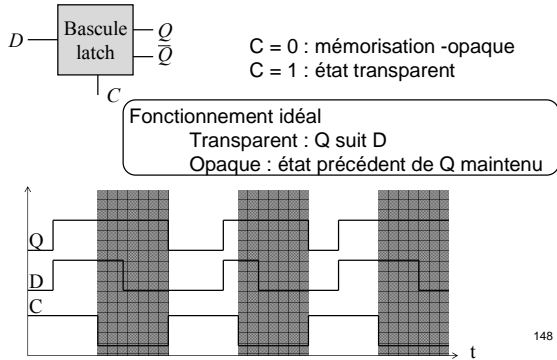
La bascule latch



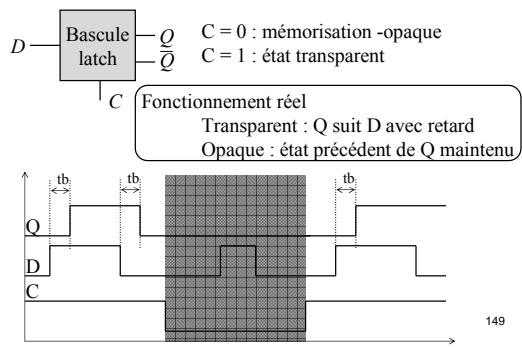
- $C = 0$: $S = R = 1$, mémorisation ; état opaque
- $C = 1$: $S = \neg D$, $R = D$, donc $Q = D$; état transparent

147

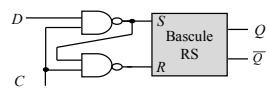
La bascule latch



La bascule latch

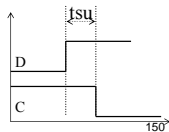


La bascule latch



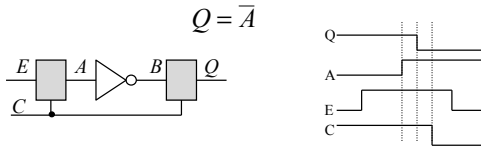
- $C = 0$: $S = R = 1$, mémorisation ; état opaque
- $C = 1$: $S = \overline{D}$, $R = D$, donc $Q = D$ et $\overline{Q} = \overline{D}$
état transparent

Temps d'établissement (setup)
 C ne doit pas repasser à 0 trop tôt
 Equivalent : D doit être stable assez longtemps avant que C passe à 0



La bascule latch

- Les données sont cohérentes sur l'état bas

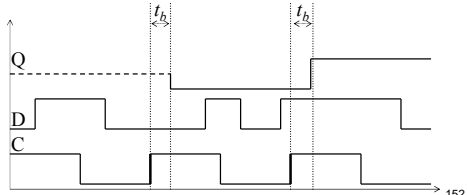


- Mais ne peut être utilisé pour $x = f(x, y)$

151

La bascule D

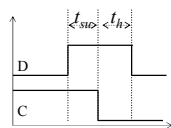
- Bascule sur front
Enregistre l'état de D sur la transition montante (0->1) de C. L'état est recopié sur Q après t_p .



152

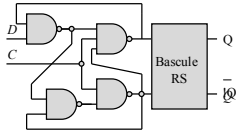
La bascule D

- Contraintes : temps d'établissement et de maintien



153

Réalisation de la bascule D

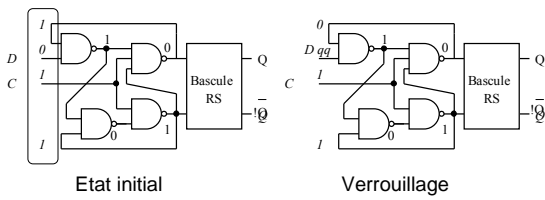


Pour $C = 0$, la bascule RS est en mémorisation

154

Réalisation de la bascule D

- Ecriture d'un 0



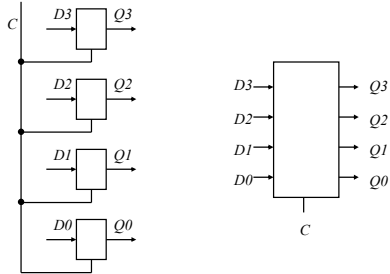
155

Registre

- Un ensemble de bascules commandées par le même signal : mémorisent simultanément n bits au lieu de 1.
 - Registre opaque : à base de bascules D
 - Registre transparent : à base de bascules latch

156

Registre



157

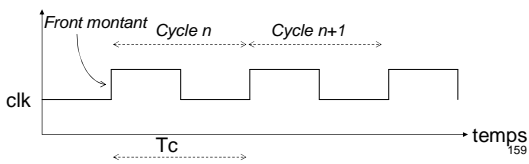
Plan

- Motivation
- Bascules et registres
Conception des circuits de mémorisation
- Automates
Utilisation des circuits de mémorisation

158

Horloge

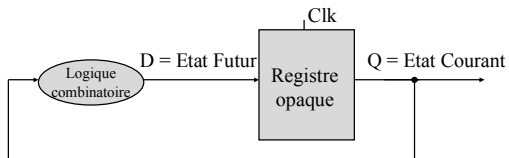
- Signal périodique. Période = Temps de cycle T_c
Fréquence $F = 1/T_c$
- Fournit une référence de temps discret commune à un ensemble de circuits



159

Compteurs synchrones

$$X_{i+1} = (1 + X_i) \bmod N$$

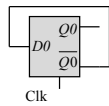


Réalisation = définir les fonctions de la logique combinatoire

163

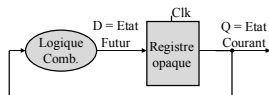
Compteur par 2 synchrone

$$D0 = \overline{Q0}$$



164

Compteur par 4



En code entier naturel

Etat Courant		Etat Futur			
Val	Q1	Q0	D1	D0	Val
0	0	0	0	1	1
1	0	1	1	0	2
2	1	1	1	1	3
3	1	0	0	0	0

$$D0 = \overline{Q0}$$

$$D1 = Q1 \cdot \overline{Q0} + \overline{Q1} \cdot Q0 = Q1 \oplus Q0$$

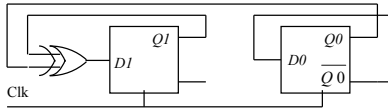
165

Compteur par 4 synchrone

$$D0 = \overline{Q0}$$

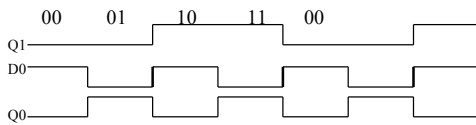
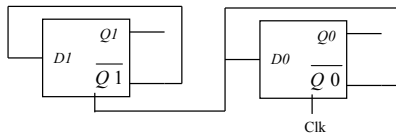
$$D1 = Q1 \cdot \overline{Q0} + \overline{Q1} \cdot Q0 = Q1 \oplus Q0$$

En utilisant les deux sorties des bascules D, on récupère directement les sorties complémentées.



166

Compteur par 4 asynchrone



Conception complexe : la commande n'est pas indépendante des bascules.

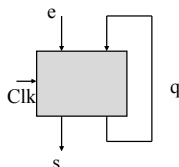
167

Automates synchrones

$$(q_{n+1}, s) = f(q_n, e)$$

- q : état de l'automate
- e : entrées
- s : sorties - commandes

Un compteur est un automate avec e absent, s = q

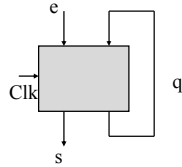


168

Application des automates synchrones

$$(q_{n+1}, s) = f(q_n, e)$$

- Circuits : de l'automate de machine à laver au contrôle de microprocesseurs
- Théorie et pratique des langages



169

Automate de Moore

Un automate de Moore est un sextuplet

$\{Q, q_0, T, E, S, f\}$

Q ensemble des états

q_0 état initial

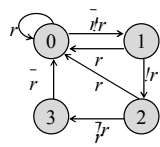
E ensemble des entrées

S ensemble des sorties

T transitions, inclus dans $Q \times E \times Q$

$f : Q \rightarrow S =$ calcul des sorties

Exemple : compteur par 4 avec RAZ



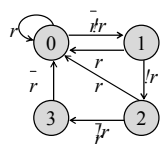
$$f(q) = q$$

170

Automate de Moore

Les sorties sont déterminées par l'état

Exemple : compteur par 4 avec RAZ

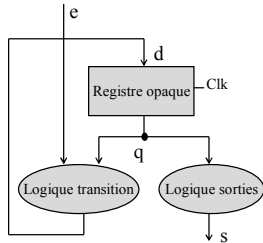


$$f(q) = q$$

171

Réalisation d'un automate de Moore

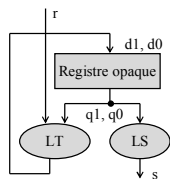
- Etat : registre opaque
- Transitions et sortie : logique combinatoire
- Réaliser un automate :
 - Taille du registre d'états
 - Synthèse des fonctions booléennes transition et sortie : PLA



172

Compteur avec RAZ

Etat courant		Etat futur				
N	q1	q0	r	d1	d0	N
0	0	0	0	0	1	1
0	0	0	1	0	0	0
1	0	1	0	1	0	2
1	0	1	1	0	0	0
2	1	0	0	1	1	3
2	1	0	1	0	0	0
3	1	1	0	0	0	0
3	1	1	1	0	0	0



$$d_0 = \overline{q_0} \cdot \overline{r}$$

$$d_1 = \overline{q_1} \cdot q_0 \cdot \overline{r} + q_1 \cdot \overline{q_0} \cdot \overline{r}$$

$$s_1 = q_1, \quad s_0 = q_0$$

173

Automate de Mealy

Un automate de Moore est un quintuplet

{Q, q0, T, E, S}

Q ensemble des états

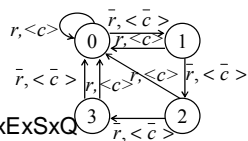
q0 état initial

E ensemble des entrées

S ensemble des sorties

T transitions, inclus dans QxExSxQ

Exemple : compteur par 4 avec RAZ et signal



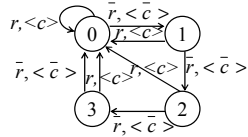
174

Automate de Mealy

T transitions, inclus dans QxExSxQ

Les sorties sont déterminées par la transition

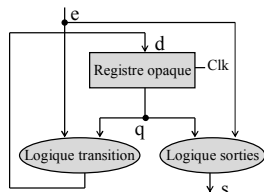
Exemple : compteur par 4 avec RAZ et signal



175

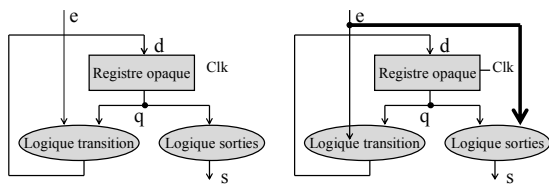
Réalisation d'un automate de Mealy

- Etat : registre opaque
- Transitions et sortie : logique combinatoire
- Réaliser un automate :
 - Taille du registre d'états
 - Synthèse des fonctions booléennes transition et sortie : PLA



176

Comparaison Moore-Mealy



177

Comparaison Moore-Mealy

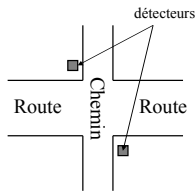
- Les deux modèles sont équivalents : étant donné un automate de Mealy (resp Moore), on peut toujours construire un automate de Moore (resp Mealy) qui a la même séquence de sorties pour une séquence d'entrées données.
- Mais l'automate de Moore a toujours au moins autant (et souvent beaucoup plus) d'états que l'automate de Mealy équivalent.

178

Exemple : contrôle de croisement

Contraintes :

- Sûr et non bloquant chemin
- Rouge Route < Temps Long
- Au moins Temps Long entre détection et Route Rouge
- Orange pendant Temps Court (exact)



Signaux :

- $a = 1$: détection (donné)
- $tl/tc = 1$: Temps Long/Court écoulé
- Contrôle des feux

179

Exemple : contrôle de croisement

Sans temporisation

RV : route vert, chemin rouge

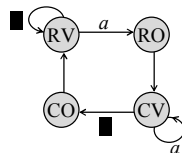
RO : route orange, chemin rouge

CV : route rouge, chemin vert

CO : route rouge, chemin orange

Seuls états possible pour la sécurité

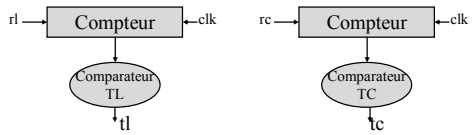
Moore : la valeur des feux est associée à l'état



180

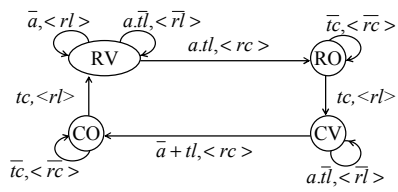
Exemple : contrôle de croisement

Avec temporisation :
Aux instants adéquats, RAZ d'un compteur



181

Exemple : contrôle de croisement

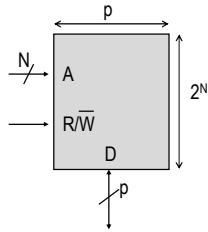


182

4. Hiérarchie Mémoire

Hierarchie mémoire

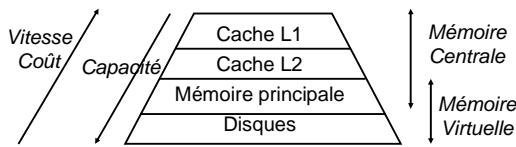
- Machine de Von-Neumann : mémoire RAM
- Un tableau linéaire (1 dimension)
- N bits d'adresse
- p bits de données
- 2^N entités de P bits sont adressables
- une entité = un mot mémoire
- Adressable = lue/écrite individuellement



184

Hierarchie mémoire

- Machine de Von-Neumann : mémoire RAM, tableau d'octets
- Réalisation matérielle



185

Mémoire statique - SRAM

- Technologie semiconducteurs logique
- 6 T par point mémoire
Un bistable plus deux interrupteurs
- Evoluent en fréquence comme les processeurs

186

Mémoire dynamique DRAM

- Technologie non CMOS
- Equivalent 1,5 transistors
- Résistance + capacité
- Demande un rafraîchissement périodique (réécriture)
- La lecture est destructrice -> Temps de cycle
- Le temps d'accès évolue moins vite que celui des processeurs

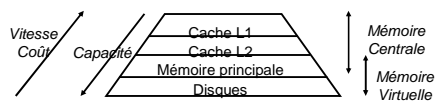
187

Faits technologiques (1)

- Gap processeur mémoire
1 calcul = 3 opérandes
- Loi de Moore: densité d'intégration X 2 / 18 mois
 - Joue à plein pour les SRAM
 - Effet plus complexe sur les processeurs
- Performance des processeurs x 1,5 / an
- Débit DRAM x 1,2 / an
- Débit disques x 1,6 / an
- Les meilleures SRAM
temps d'accès = T_c processeur
- 1 ordre de grandeur SRAM -> DRAM
- 3 ordres de grandeur DRAM -> disques

188

Hierarchie mémoire

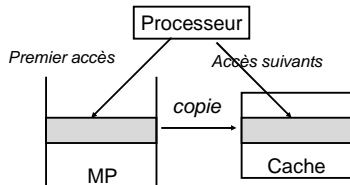


- A un instant donné, chaque niveau de la hiérarchie contient un extrait du niveau inférieur
- L'accès étant plus rapide, la copie offre l'illusion d'une mémoire + rapide à faible coût
- La gestion des copies et problèmes reliés doit être transparente au niveau de l'exécution des instructions
 - Cache : entièrement en matériel
 - Mémoire virtuelle : matériel et logiciel système

189

Le principe de localité

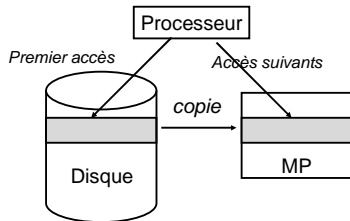
- Localité temporelle = Accès répétés à une même adresse
 - Boucles dans le code
 - Éventuellement en données



190

Le principe de localité

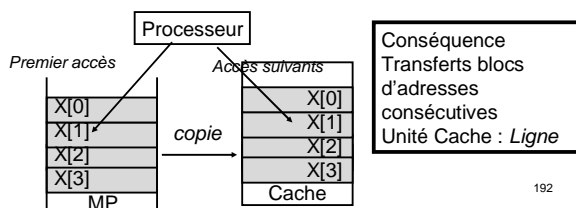
- Localité temporelle = Accès répétés à une même adresse
 - Boucles dans le code
 - Éventuellement en données



191

Le principe de localité

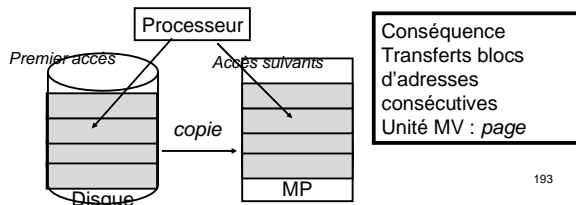
- Localité spatiale
 - Accès à des adresses consécutives
 - Le coût d'un accès bloc est beaucoup moindre que le coût de n accès indépendants : $T(\text{bloc}) = T_s + n T_i$



192

Le principe de localité

- Localité spatiale
 - Accès à des adresses consécutives
 - Le coût d'un accès bloc est beaucoup moindre que le coût de n accès indépendants : $T(\text{bloc}) = T_s + n T_i$



Le produit Matrice-Vecteur

194

Le produit Matrice-Matrice

195

Conséquences pour le programmeur

- Attention aux conventions des compilateurs

```
pour i= 1,N  
... A[i] ...  
fin pour
```

Bon

- Les accès par indirection peuvent coûter TRES cher

```
pour i= 1,N  
... A[L[i]]  
...  
fin pour
```

Mauvais

196

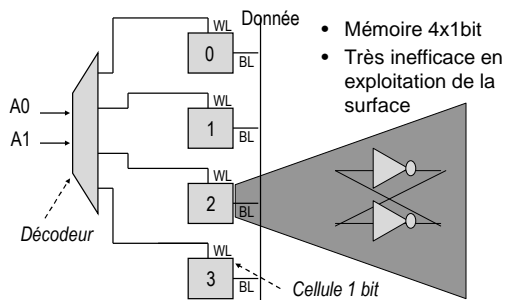
Conséquences pour le programmeur

Des gains de performances importants peuvent être obtenus en localisant les accès

- Il existe des outils d'analyse des performances processeur et mémoire
 - VTune pour Windows
 - Associés à des compilateurs commerciaux sous Linux/Unixes
 - Quelques logiciels libres produits de recherche
- L'optimisation automatique est un sujet de recherche

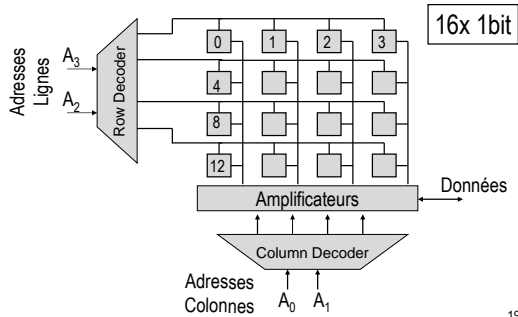
197

Organisation interne : 1D



198

Organisation interne matricielle



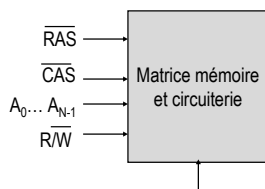
199

Organisation matricielle

- L'organisation matricielle concerne une mémoire $2^{2m} \times 1$ bit
- Pour une mémoire $2^{2m} \times p$ bits, la structure est répliquée

200

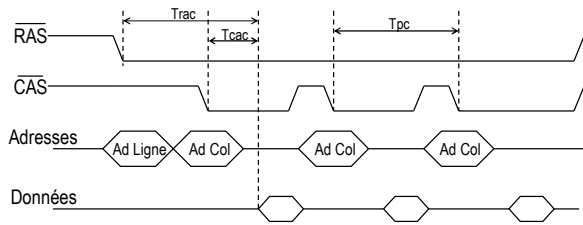
Organisation des DRAM



- !RAS et !CAS valident respectivement les parties hautes et basses des adresses

201

Chronogramme en lecture



202

Organisation matricielle + validation Justifie la localité spatiale

- Les données d'une même ligne sont présentées simultanément, puis sélectionnées
- Possibilité d'accès consécutifs aux données d'une même ligne *plus rapides* qu'à ceux de lignes distinctes

203
