

Circuits logiques

Eric Cariou

*Université de Pau et des Pays de l'Adour
Département Informatique*

Eric.Cariou@univ-pau.fr

Circuit logique

- ◆ Circuit électronique réalisant une ou plusieurs fonctions logiques
- ◆ Un circuit logique est composé
 - ◆ D'un ensemble de portes logiques
 - ◆ De circuits logiques
 - ◆ Le tout interconnectés entre eux
- ◆ 2 types de circuits logiques
 - ◆ Circuits combinatoires : $S = f(E)$
 - ◆ Circuits séquentiels : notion d'état et de mémoire

Circuits Logiques

(1) Circuits Combinatoires

Circuit combinatoire

- ◆ $S_j = f(E_i)$
 - ◆ Les sorties S_j sont fonctions uniquement de la valeur des entrées E_i
- ◆ Un circuit combinatoire est défini par une ou plusieurs fonctions logiques
 - ◆ Définition de la valeur des sorties en fonction des entrées du circuit
 - ◆ Algèbre de Boole et les fonctions logiques sont donc le support théorique des circuits combinatoires
- ◆ Un circuit se représente par un logigramme

Portes logiques

- ◆ Une porte logique
 - ◆ Circuit combinatoire de base
 - ◆ Réalisant une opération logique de base
 - ◆ Exemple : OU, ET, NON, correspondant aux opérateurs de l'algèbre de Boole
- ◆ Une porte possède
 - ◆ Une table de vérité et/ou une expression logique définissant son résultat en fonction de son/ses entrée(s)
 - ◆ Un symbole graphique

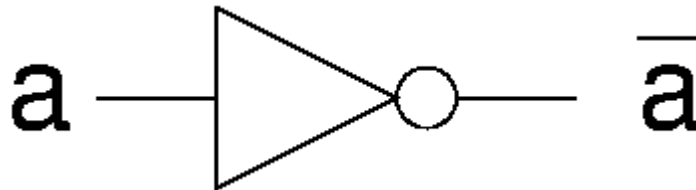
Porte NON

- ◆ Porte NON (NOT)

- ◆ 1 entrée, 1 sortie

- ◆ *NON* a est noté \bar{a}

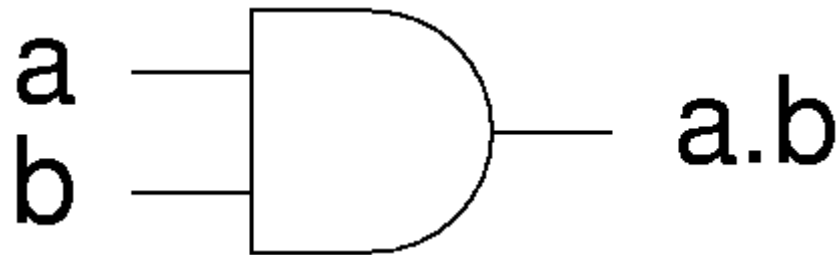
a		\bar{a}
0		1
1		0



Porte ET

- ◆ Porte ET (AND)
- ◆ 2 entrées, 1 sortie
- ◆ a ET b est noté $a.b$ ou ab ou $a \wedge b$

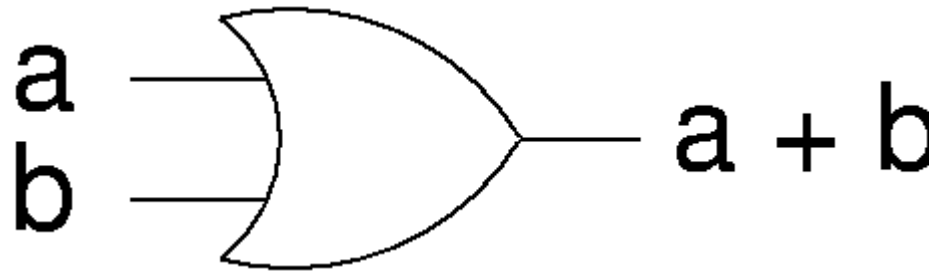
a	b		a.b
0	0		0
0	1		0
1	0		0
1	1		1



Porte OU

- ◆ Porte OU (OR)
- ◆ 2 entrées, 1 sortie
- ◆ a OU b est noté $a + b$ ou avb

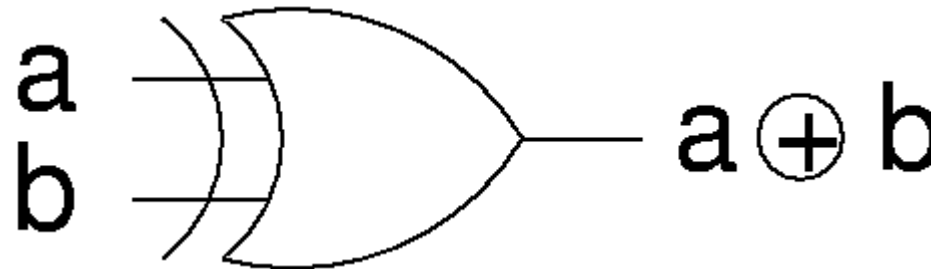
a	b		a + b
0	0		0
0	1		1
1	0		1
1	1		1



Porte OU exclusif

- ◆ Porte OU-exclusif (XOR)
- ◆ 2 entrées, 1 sortie
- ◆ a OU-exclusif b est noté $a \oplus b$

a	b		$a \oplus b$
0	0		0
0	1		1
1	0		1
1	1		0



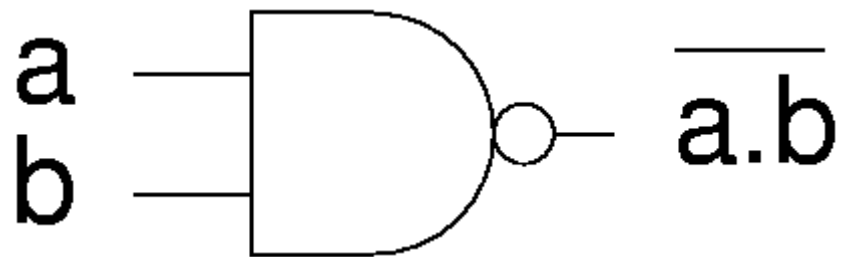
Porte NON ET

- ◆ Porte NON ET (NAND)

- ◆ 2 entrées, 1 sortie

- ◆ a NAND b est noté $\overline{a \cdot b}$

a	b		$\overline{a \cdot b}$
0	0		1
0	1		1
1	0		1
1	1		0



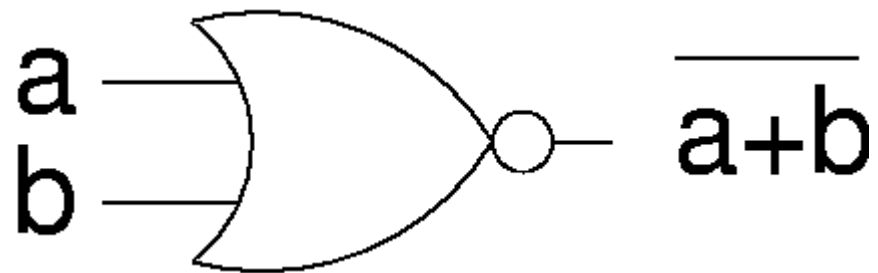
Porte NON OU

- ◆ Porte NON OU (NOR)

- ◆ 2 entrées, 1 sortie

- ◆ a NOR b est noté $\overline{a + b}$

a	b		$\overline{a + b}$
0	0		1
0	1		0
1	0		0
1	1		0



Autres portes

- ◆ Pour chaque porte à 2 entrées
 - ◆ Variantes à 3, 4, ... entrées (mais toujours une seule sortie)
 - ◆ Généralisation de la fonction logique de base à plus de 2 variables en entrée
 - ◆ Le symbole graphique utilisé est identique mais avec plus de 2 entrées
- ◆ Exemples
 - ◆ Porte ET à 3 entrées a , b et c a pour expression logique : $a.b.c$
 - ◆ Porte NOR à 4 entrées a , b , c et d a pour expression logique : $\overline{a + b + c + d}$

Synthèse d'un circuit logique

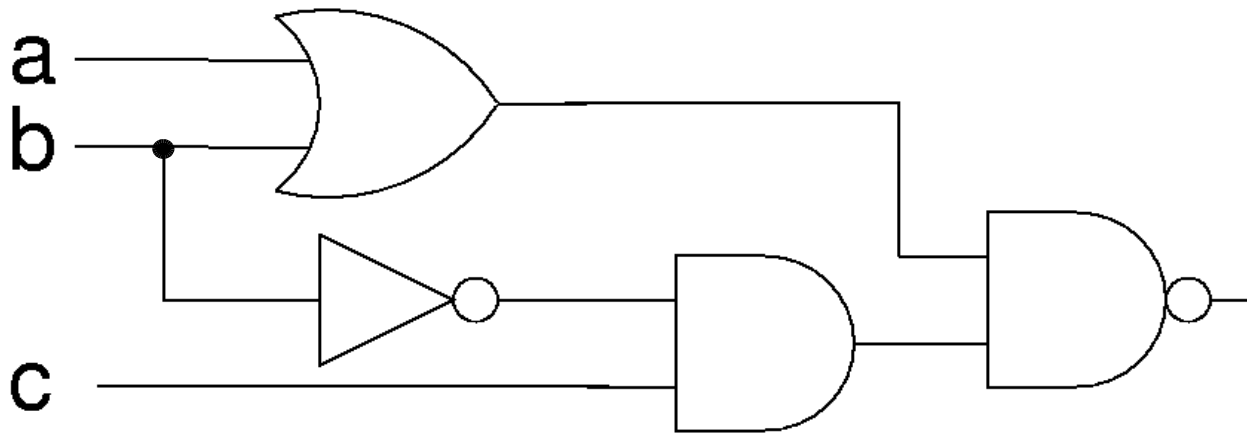
- ◆ A partir d'une fonction logique
 - ◆ Trouver le logigramme correspondant à cette fonction
- ◆ Principe
 - ◆ Simplifier la fonction logique avec 2 méthodes
 - ◆ La méthode algébrique (algèbre de Boole)
 - ◆ La méthode des tableaux de Karnaugh
 - ◆ En déduire le logigramme correspondant

Analyse de circuit logique

- ◆ A partir du logigramme d'un circuit
 - ◆ Trouver sa fonction logique
- ◆ Principe
 - ◆ Donner l'expression des sorties de chaque porte/composant en fonction des valeurs de ses entrées
 - ◆ En déduire au final la (ou les) fonction(s) logique(s) du circuit
- ◆ On peut ensuite
 - ◆ Déterminer la table de vérité du circuit
 - ◆ Simplifier la fonction logique à l'aide des propriétés de l'algèbre de Boole ou les tableaux de Karnaugh

Exemple d'analyse de circuit

- ◆ Exemple de circuit logique
 - ◆ 3 entrées, 1 sortie
 - ◆ Composé uniquement de portes logiques



Exemple d'analyse de circuit

- ◆ Quelle est la fonction logique de ce circuit ?
 - ◆ A partir de son logigramme
 - ◆ $f(a, b, c) = \overline{(a + b) \cdot (\bar{b} \cdot c)}$
 - ◆ Après simplification
 - ◆ $f(a, b, c) = \bar{a} + b + \bar{c}$
 - ◆ En nombre de portes minimales
 $f(a, b, c) = \overline{ac} + b$

Exemple d'analyse de circuit

◆ Table de vérité de l'exemple

a	b	c	a+b=x	\bar{b}	$\bar{b}.c=y$	x.y	$\overline{x.y}$
0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1
0	1	0	1	0	0	0	1
0	1	1	1	0	0	0	1
1	0	0	1	1	0	0	1
1	0	1	1	1	1	1	0
1	1	0	1	0	0	0	1
1	1	1	1	0	0	0	1

Exemple d'analyse de circuit

- ◆ Table de vérité de la fonction simplifiée

a	b	c		\bar{a}		\bar{c}		$\bar{a}+b+\bar{c}$
0	0	0		1		1		1
0	0	1		1		0		1
0	1	0		1		1		1
0	1	1		1		0		1
1	0	0		0		1		1
1	0	1		0		0		0
1	1	0		0		1		1
1	1	1		0		0		1

- ◆ On trouve les mêmes valeurs dans les 2 tables
 - ◆ Les fonctions sont bien égales

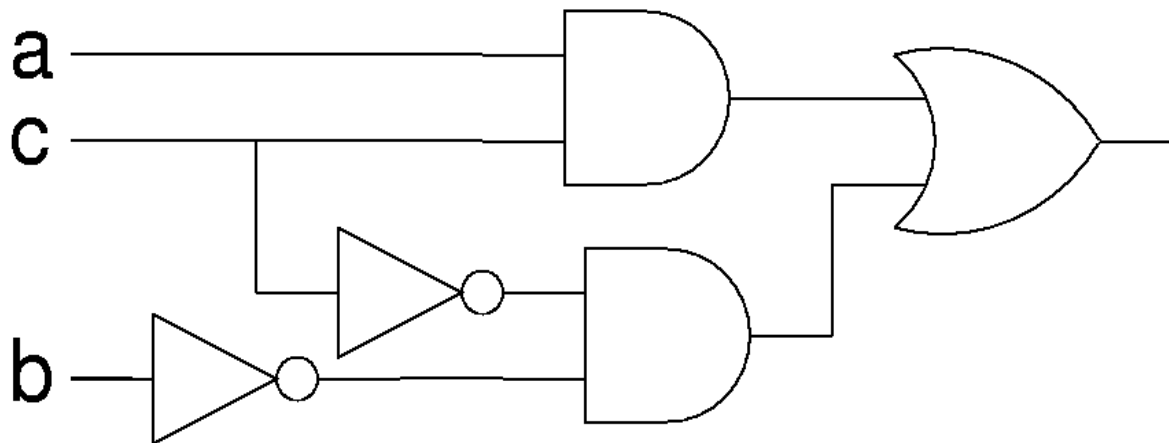
Exemple de synthèse de circuit

- ◆ Soit la fonction

$$f(a, b, c) = abc + a\bar{b}\bar{c} + \bar{a}\bar{b}\bar{c} + a\bar{b}c$$

- ◆ Après simplification, on obtient

$$f(a, b, c) = ac + \bar{b}\bar{c}$$



Additionneur demi-bit

- ◆ Exemple de circuit : un additionneur demi-bit
 - ◆ Réalise l'addition de 2 nombres codés chacun sur 1 bit
 - ◆ Doit pouvoir gérer l'éventuel débordement
 - ◆ Table de vérité du circuit avec :

- ◆ S : la somme des 2 bits

- ◆ R : la retenue

x	y	S	R
+ - - - +	+ - - - +	+ - - - +	+ - - - +
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

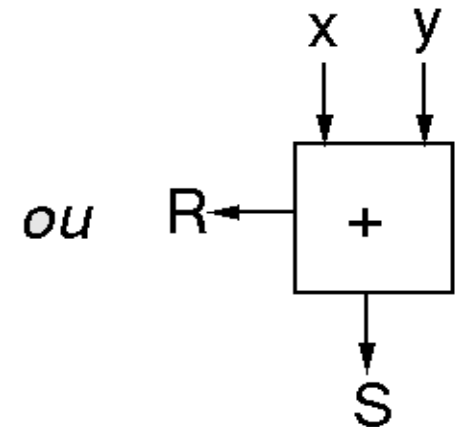
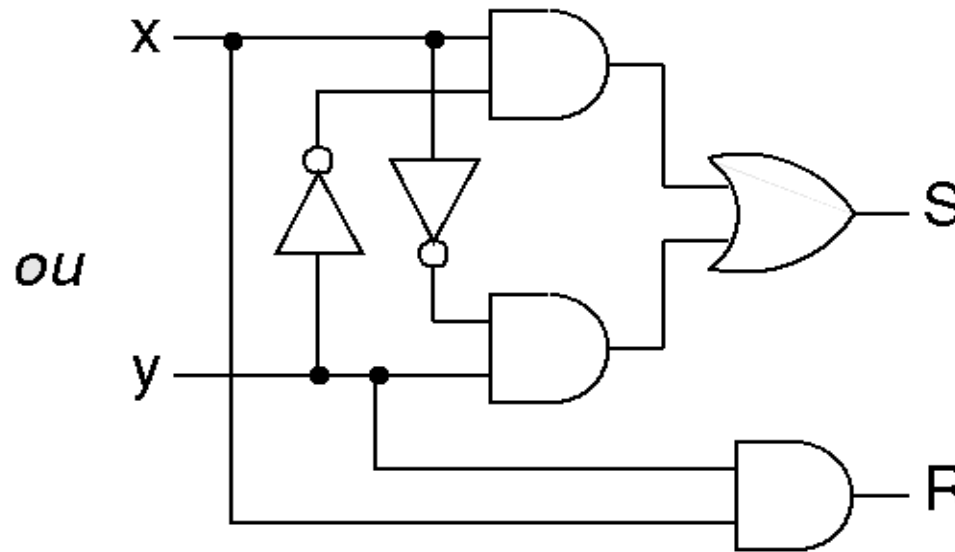
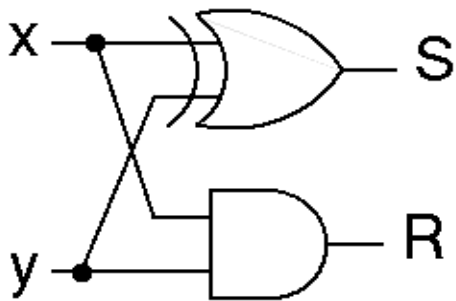
- ◆ Note : on parle de « demi-bit » car ne prend pas en compte une retenue en entrée (voir suite)

Additionneur demi-bit

◆ Fonctions logiques de S et R

◆ $S = x\bar{y} + \bar{x}y = x \oplus y$

◆ $R = xy$



Additionneur n bits

- ◆ Additionneur demi-bit
 - ◆ Fait une somme de 2 bits
- ◆ Additionneur n bits
 - ◆ Fait l'addition de 2 nombres de n bits
 - ◆ Deux réalisations possibles
 - ◆ Concevoir entièrement l'additionneur à partir de rien
 - ◆ Concevoir l'additionneur en réutilisant d'autres composants : des additionneurs 1 bit

Additionneur n bits

◆ Additionneur n bits

- ◆ Si on veut le construire à partir de n additionneurs 1 bit
- ◆ Doit prendre en compte les retenues intermédiaires générées et leur propagation

1 - - > *retenue générée par l'addition des chiffres*
010 *précédents à additionner en plus aux 2 chiffres*
+111
- - - - -
1001

◆ Additionneur demi-bit précédent

- ◆ Insuffisant : prend seulement 2 bits en entrée, pas la retenue
- ◆ On parlait donc de demi-additionneur
- ◆ Additionneur qui prend en compte une retenue en entrée :
additionneur 1 bit complet

Additionneur 1 bit complet

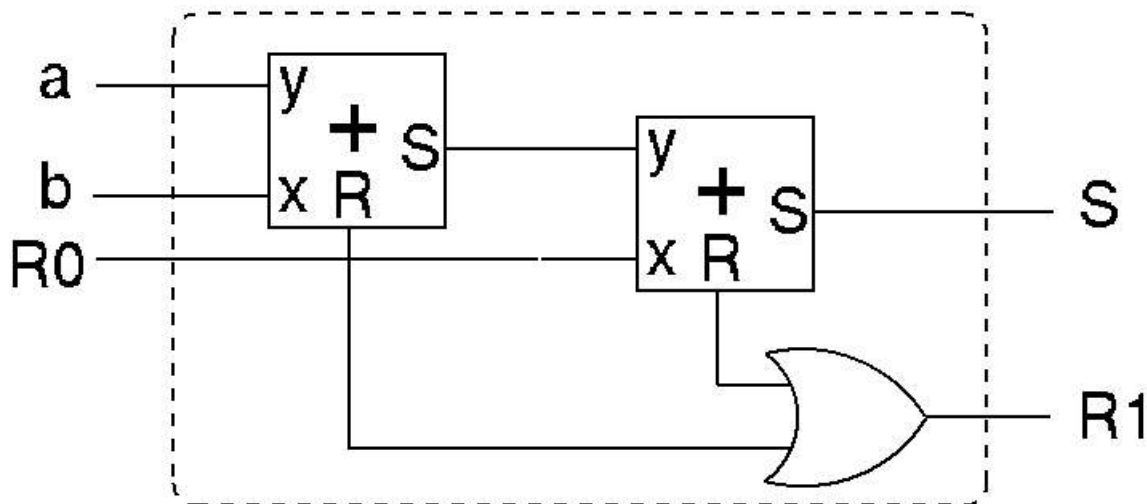
- ◆ Additionneur 1 bit complet
 - ◆ Entrées : bits x , y et retenue R_0
 - ◆ Sorties : résultat S et retenue R_1
- ◆ 2 idées pour définir l'additionneur 1 bit complet
 - ◆ Déterminer la table de vérité de $\{S, R_1\} = f(x, y, R_0)$
 - ◆ On obtient un nouveau circuit, plus complexe
 - ◆ Utiliser le demi-additionneur vu précédemment
 - ◆ On réutilise le circuit déjà défini

Additionneur 1 bit complet

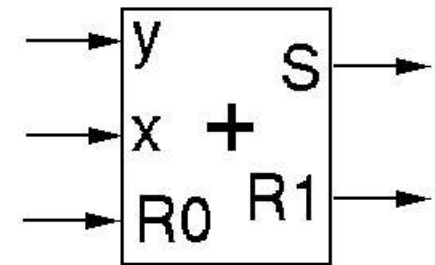
- ◆ En réutilisant le demi-additionneur
 - ◆ Le calcul de S se fait via 2 additions
 - ◆ $x + y = z$ et $S = z + R_0$
 - ◆ Chaque addition est effectuée par un demi-additionneur
 - ◆ La retenue finale R_1 est le OU des 2 retenues générées par les 2 additions
 - ◆ Si la première addition génère une retenue : on aura forcément une retenue au final
 - ◆ Si la seconde addition génère une retenue : on aura aussi une retenue au final
 - ◆ Donc on fait le OU des 2 retenues

Additionneur 1 bit complet

- ◆ Additionneur 1 bit complet
 - ◆ S : résultat de l'addition des bits a et b et de la retenue R_0
 - ◆ $R1$: retenue générée

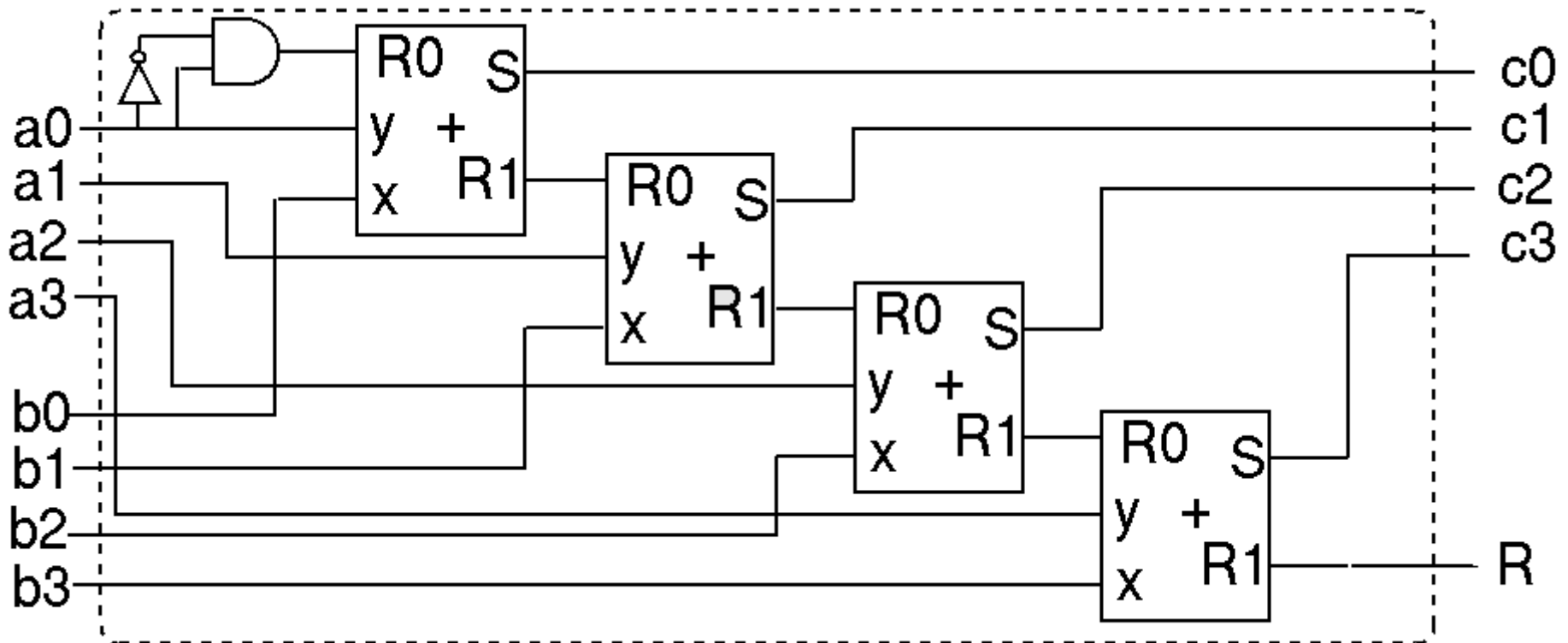


ou



Additionneur n bits

- ◆ Exemple pour additionneur 4 bits
 - ◆ On enchaîne en série 4 additionneurs 1 bit complet
 - ◆ Le résultat est connu après propagation des valeurs calculées le long de tout le circuit
 - ◆ $C = A + B$, en précision 4 bits. R : retenue globale



Ensembles complets de fonctions logiques de base

- ◆ Dans les exemples précédents
 - ◆ On a construit des circuits avec n'importe quelles portes : ET, OU, NON, NAND, NOR, XOR ...
- ◆ Il existe des ensembles complets de portes (fonctions logiques de base)
 - ◆ Les portes d'un ensemble complet permettent de réaliser n'importe quel circuit (fonction logique)
 - ◆ Peut être utile pour limiter le coût d'un circuit
 - ◆ Le nombre de transistors utilisés pour réaliser une porte peut varier du simple au double selon les portes

Ensembles complets de fonctions logiques de base

- ◆ 3 ensembles complets de fonctions
 - ◆ {ET, OU, NON}
 - ◆ {NOR}
 - ◆ {NAND}
- ◆ Pour prouver que ces ensembles sont complets
 - ◆ Montrer qu'on peut spécifier toutes les fonctions de base à partir des fonctions d'un ensemble

Ensembles complets de fonctions logiques de base

- ◆ Ensembles complets minimaux : contenant le moins d'éléments
- ◆ {ET, OU, NON} : pas minimal car
 - ◆ $a + b = \overline{\overline{a} \cdot \overline{b}}$: OU se définit avec ET et NON
 - ◆ $a \cdot b = \overline{\overline{a} + \overline{b}}$: ET se définit avec OU et NON
- ◆ {ET, NON} et {OU, NON} sont 2 ensembles complets mais on ne peut pas les simplifier plus

Ensembles complets de fonctions logiques de base

- ◆ Au final : 2 ensembles complets minimaux
 - ◆ {NAND} et {NOR}
- ◆ Preuve pour NAND
 - ◆ On doit pouvoir définir à partir d'un NAND :
 - ◆ NON : $\text{NON}(a) = \text{NAND}(a,a)$ $\bar{a} = \overline{a \cdot a}$
 - ◆ OU : $\text{OU}(a,b) = \overline{\overline{\text{NAND}(a,a)} \cdot \overline{\text{NAND}(b,b)}}$
 $a + b = \overline{\bar{a} \cdot \bar{a} \cdot \bar{b} \cdot \bar{b}}$
 - ◆ ET : $\text{ET}(a,b) = \overline{\overline{\text{NAND}(a,b)} \cdot \overline{\text{NAND}(a,b)}}$
 $a \cdot b = \overline{\bar{a} \cdot \bar{b} \cdot a \cdot b}$
 - ◆ NOR, XOR : combinaisons de NON, ET et OU donc peut s'exprimer avec des NAND

Simplification des fonctions/circuits

- ◆ Pour une fonction logique, pour une question de lisibilité, on préfère la version
 - ◆ Avec les termes les plus simples
 - ◆ Reliés par le moins d'opérateurs
- ◆ En pratique, lors de la réalisation du circuit
 - ◆ Ca ne sera pas forcément la version la moins coûteuse en terme d'éléments électroniques (transistors)
 - ◆ Une porte NAND ou NOR est moins coûteuse en transistors qu'une porte OU ou ET

Simplification des fonctions/circuits

- ◆ Pour un circuit logique, on cherchera donc à le réaliser avec des portes NAND ou NOR
 - ◆ Permet de diminuer le nombre total de transistors
 - ◆ Permet de ne pas multiplier le nombre de portes différentes à utiliser
- ◆ Exemple : demi-additionneur 1 bit
 - ◆ $S(x, y) = x\bar{y} + \bar{x}y$ et $R = xy$
 - ◆ On va exprimer R et S avec des NAND
 - ◆ On utilise également des NON pour simplifier

Demi-additionneur avec NAND

◆ Pour transformer R et S, on passe par l'involution

◆ $S(x, y) = x\bar{y} + \bar{x}y$

$$\overline{S(x, y)} = \overline{x\bar{y} + \bar{x}y} = \overline{x\bar{y}}\overline{\bar{x}y}$$

◆ $R = xy = \overline{\overline{xy}} = \overline{(\overline{xy})}$

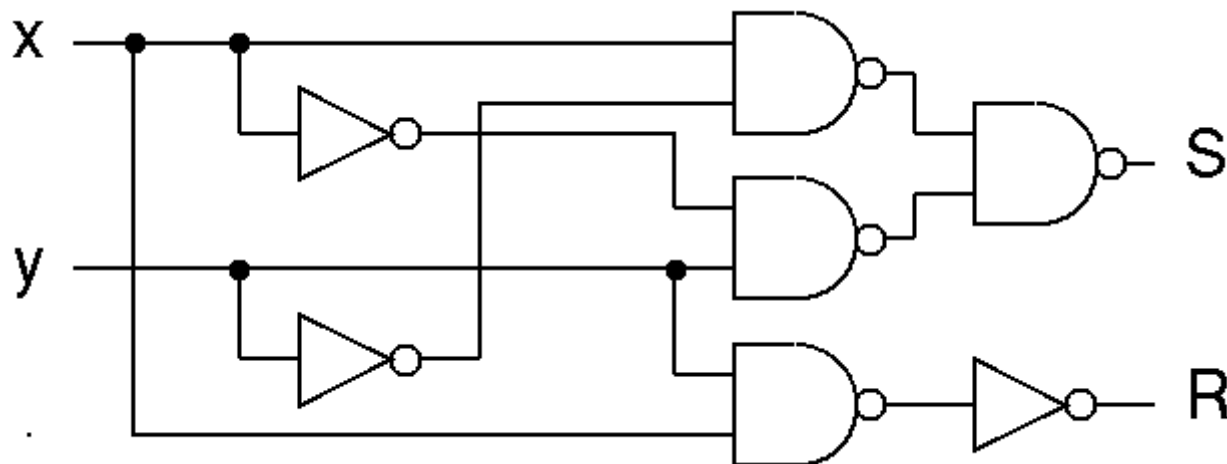


Table de vérité -> circuit en portes NAND

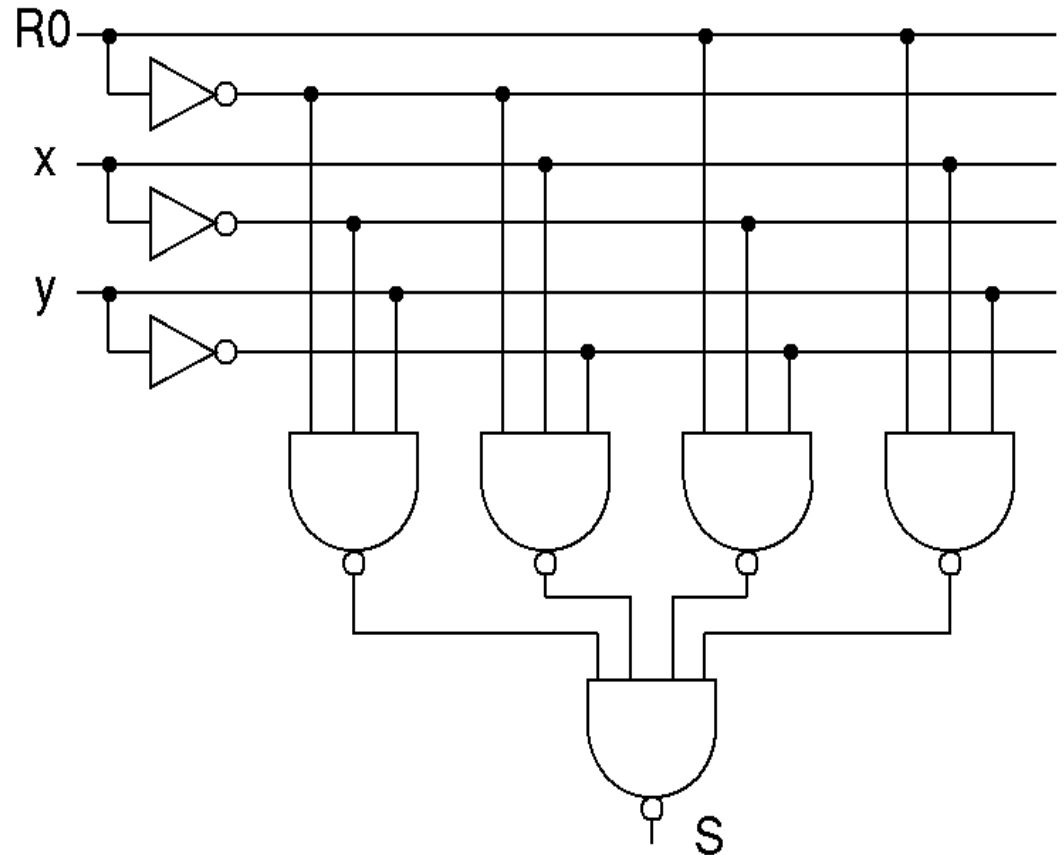
- ◆ Méthode pour passer de la table de vérité au circuit réalisé avec des NAND (et des ET)
- ◆ 2 couches de portes NAND
 - ◆ Première couche :
 - ◆ Pour chaque valeur de $f(X_i)$ égale à 1
 - ◆ On fait un NAND de tous les X_i en prenant X_i si $X_i = 1$ ou $\overline{X_i}$ si $X_i = 0$
 - ◆ Deuxième couche : on fait un NAND de toutes les sorties des NAND de la première couche
 - ◆ Nécessite des portes NAND a plus de 2 entrées

Table de vérité -> circuit en portes NAND

◆ Exemple : avec additionneur complet 1 bit

◆

R0	x	y	S	R1
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



◆ Avantage de la méthode : construction systématique

◆ Inconvénient : pas toujours optimal en nb de portes 36

Logique à 3 états

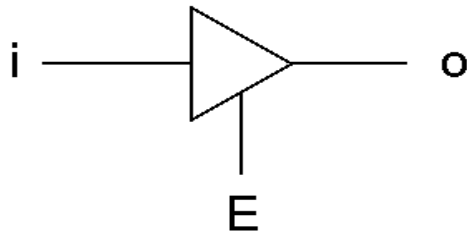
- ◆ Algèbre de Boole et circuits logiques : logique à 2 états
 - ◆ Etat 0 (état « faux » ou état physique « courant électrique nul »)
 - ◆ Etat 1 (état « vrai » ou état physique « courant électrique non nul »)
- ◆ Logique à 3 états
 - ◆ Extension de la logique à 2 états avec un troisième état : « état indéfini »
 - ◆ Utilité principale : activer ou désactiver des parties d'un circuit liées aux mêmes sorties ou éléments

Logique à 3 états

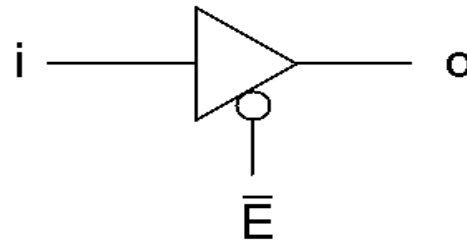
- ◆ Une porte, un élément de circuit de logique à 3 états possède une entrée supplémentaire
 - ◆ E : Enable
 - ◆ Si $E = 1$, alors les sorties de cette porte/élément sont actives et ont une valeur de 0 ou 1
 - ◆ Si $E = 0$, alors les sorties ne sont pas activées et n'ont pas de signification, comme si les sorties étaient « déconnectées »
 - ◆ \bar{E} : variante complémentée
 - ◆ Activé à 0 et désactivé à 1
- ◆ On peut connecter deux sorties ensemble, seule celle qui est activée positionnera la valeur (0 ou 1) du conducteur connectant ces deux sorties
 - ◆ Multiplexage (voir suite)

Logique à 3 états

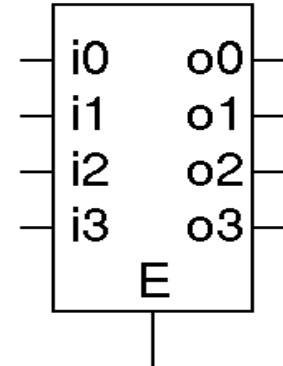
- ◆ Exemples de portes/composants de logique à 3 états



(1)



(2)



(3)

- ◆ Portes (1) et (2)

- ◆ Éléments les plus simples : active ou désactive la sortie o selon la valeur de E

- ◆ Pour (1) : si $E = 1$ alors $o = i$, si $E = 0$, alors $o = ?$ (indéfini)

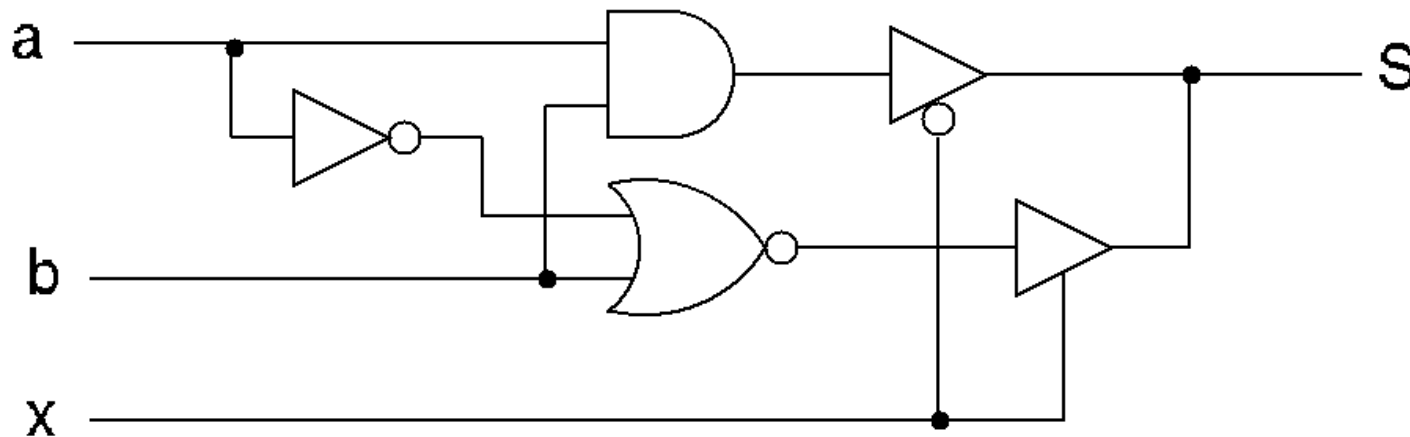
- ◆ Pour (2) : si $E = 0$ alors $o = i$, si $E = 1$, alors $o = ?$ (indéfini)

- ◆ Porte (3) : active les 4 sorties en fonction de E

- ◆ Si $E = 1$, alors chaque $o_x = i_x$ pour tous les x , sinon tous les o_x sont indéfinis

Logique à 3 états

◆ Exemple de circuit



◆ Selon la valeur de x , S correspond à la sortie d'une des 2 portes

- ◆ Si $x = 0$ alors $S = a b$
- ◆ Si $x = 1$ alors $S = \overline{a + b} = a \overline{b}$

Circuits logiques de base

- ◆ Dans beaucoup de circuits, on retrouvera certaines fonctionnalités/composants logiques
 - ◆ Additionneur 1 bit complet et additionneur n bits
 - ◆ Multiplexeur : une des X entrées vers 1 sortie
 - ◆ Démultiplexeur : 1 entrée vers une des X sorties
 - ◆ Décodeur : active une des X sorties selon un code en entrée
 - ◆ Codeur : pour 1 entrée active, fournit un code
 - ◆ Transcodeur : pour un code A fournit un code B

Multiplexeur

- ◆ X entrées et 1 sortie
- ◆ Selon une adresse, la sortie prend la valeur d'une des X entrées
- ◆ Circuit combinatoire avec
 - ◆ Une sortie K
 - ◆ Une adresse codée sur n bits
 - ◆ 2^n entrées k_x

Multiplexeur à 4 entrées

- ◆ 4 entrées, adresse sur 2 bits : a et b

- ◆ Table de vérité

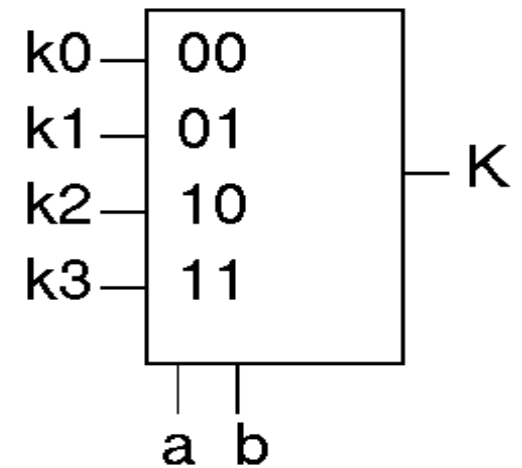
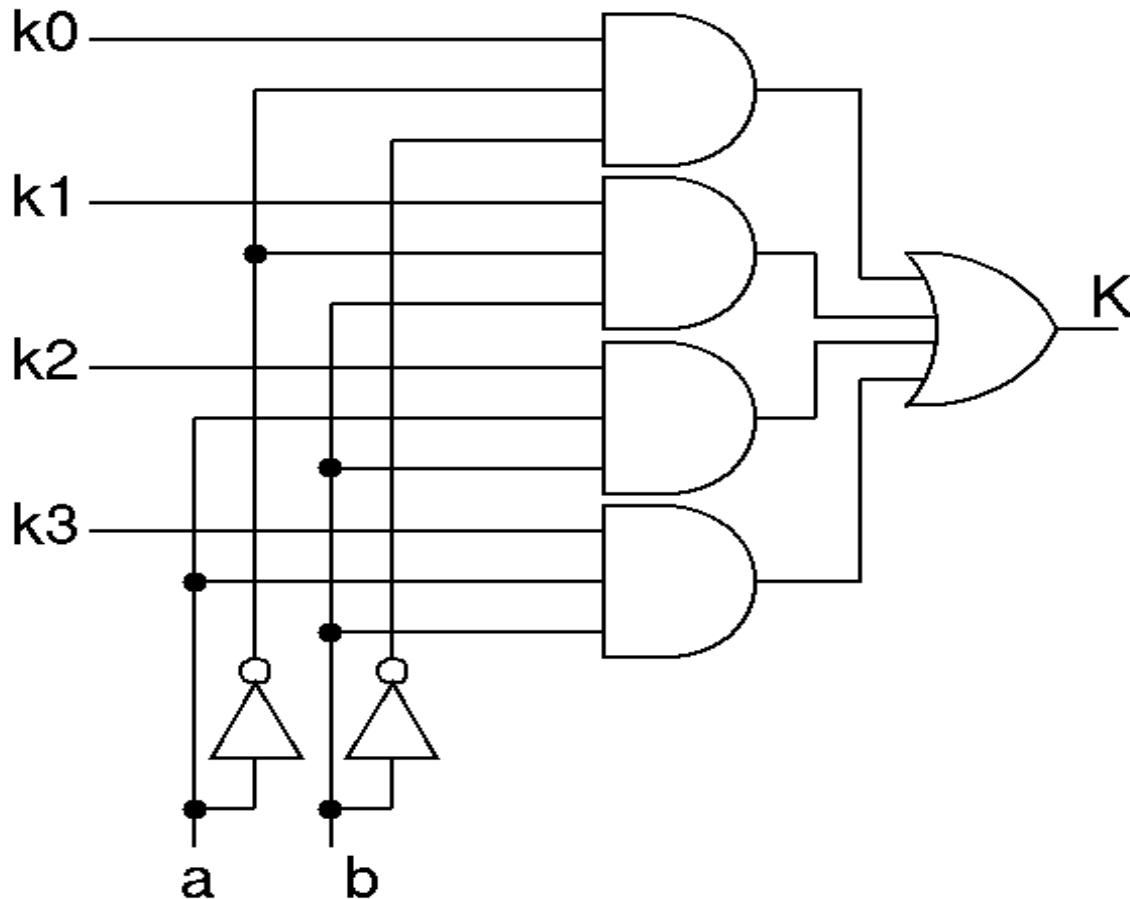
a	b	K	
+ - - -	+ - - -	+ - - -	+
0	0	k ₀	
0	1	k ₁	
1	0	k ₂	
1	1	K ₃	

- ◆ $K(a, b) = k_0 \bar{a} \bar{b} + k_1 \bar{a} b + k_2 a \bar{b} + k_3 a b$

- ◆ D'autres choix de multiplexage sont possibles

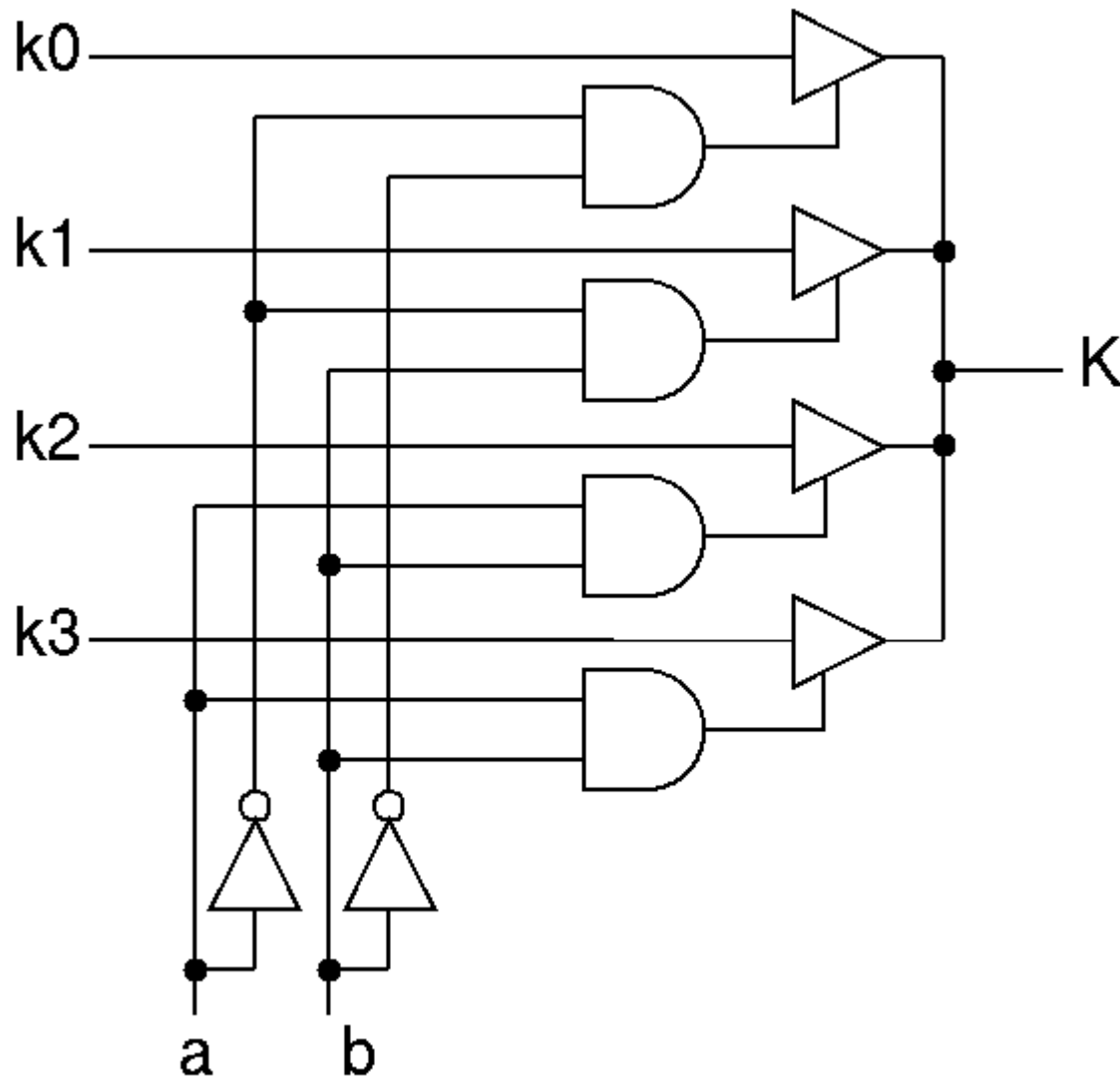
Multiplexeur 4 entrées

- ◆ Logigramme et symbole pour le multiplexeur à 4 entrées



Multiplexeur 4 entrées

◆ Variante avec logique à 3 états



◆ Selon la valeur de a et b , on ne redirige qu'un des quatre k_x vers K

Démultiplexeur

- ◆ 1 entrée, X sorties
- ◆ Selon une adresse, une des X sorties prend la valeur de l'entrée
- ◆ Circuit combinatoire avec
 - ◆ 2^n sorties k_x
 - ◆ 1 entrée K
 - ◆ Une adresse codée sur n bits

Démultiplexeur à 4 sorties

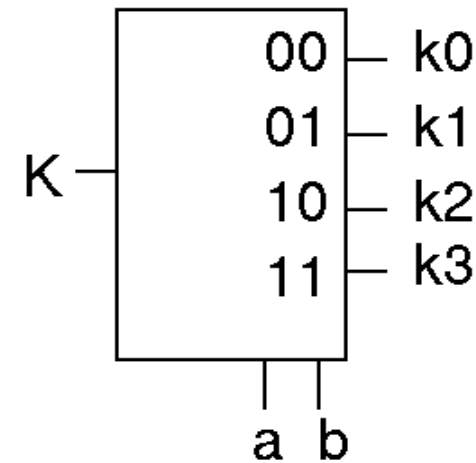
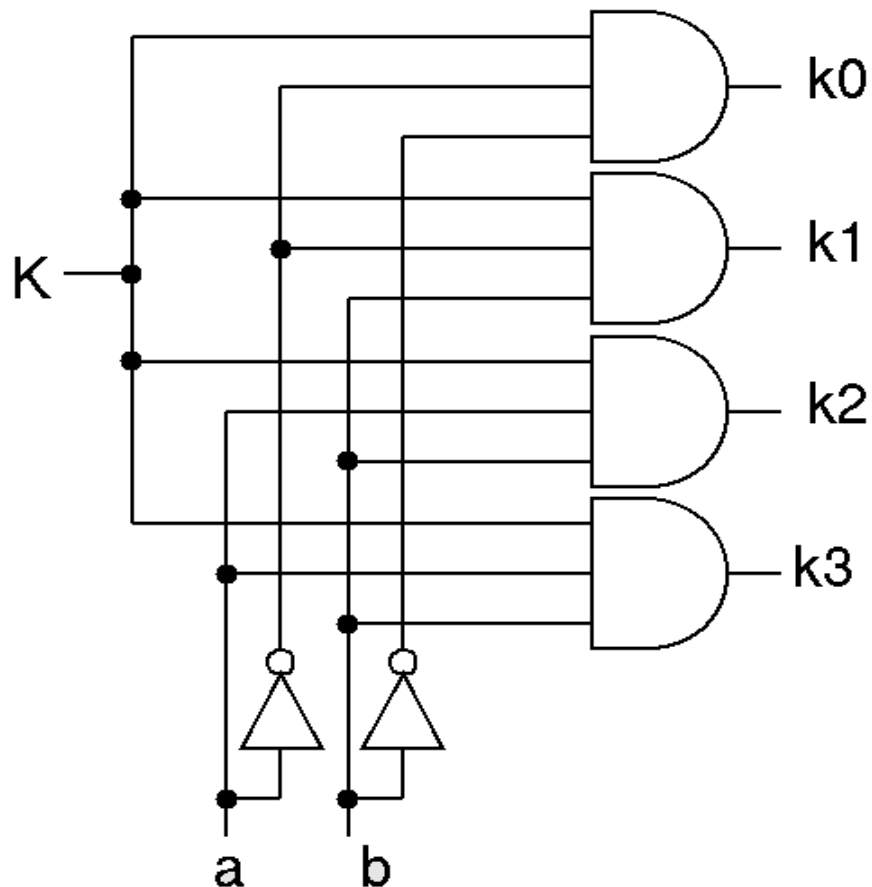
- ◆ 4 sorties, adresse sur 2 bits : a et b
- ◆ Valeurs des k_x sorties selon a et b

a	b	k_0	k_1	k_2	k_3
0	0	K	0	0	0
0	1	0	K	0	0
1	0	0	0	K	0
1	1	0	0	0	K

- ◆ $k_0 = \bar{a} \bar{b} K$ $k_1 = \bar{a} b K$
 $k_2 = a \bar{b} K$ $k_3 = a b K$

Démultiplexeur à 4 sorties

- ◆ Logigramme et symbole pour le démultiplexeur à 4 sorties



Codeur

- ◆ Active un code selon l'une des X entrées actives
 - ◆ 2^n (en général) entrées
 - ◆ 1 entrée active (valeur 1)
 - ◆ Les autres sont toutes désactivées (valeur 0)
 - ◆ Code en sortie : sur n bits
- ◆ Exemple classique de codeur
 - ◆ Numérotation de 0 à $2^n - 1$ des entrées
 - ◆ Le code représente le numéro de l'entrée codé en binaire

Codeur sur 3 bits

- ◆ 3 bits S_x en sortie et 8 entrées E_y

E0	E1	E2	E3	E4	E5	E6	E7		S0	S1	S2
1									0	0	0
	1								1	0	0
		1							0	1	0
			1						1	1	0
				1					0	0	1
					1				1	0	1
						1			0	1	1
							1		1	1	1

Décodeur

- ◆ Active une des X sorties selon un code
 - ◆ Code : sur n bits
 - ◆ Nombre de sorties : 2^n (en général)
- ◆ Exemple classique de décodeur
 - ◆ Numérotation de 0 à $2^n - 1$ des sorties
 - ◆ Le code représente le numéro codé en binaire de la sortie à activer

Décodeur sur 3 bits

- ◆ 3 bits S_x : 8 sorties E_y

S2	S1	S0		E0	E1	E2	E3	E4	E5	E6	E7
0	0	0		1							
0	0	1			1						
0	1	0				1					
0	1	1					1				
1	0	0						1			
1	0	1							1		
1	1	0								1	
1	1	1									1

Transcodeur

- ◆ Fait correspondre un code A en entrée sur n lignes à un code B en sortie sur m lignes
- ◆ Exemple avec $n = 3$ et $m = 5$

a2	a1	a0		b4	b3	b2	b1	b0
0	0	0		1	0	0	1	0
0	0	1		0	1	0	0	1
0	1	0		1	0	1	0	0
0	1	1		0	1	0	1	0
1	0	0		0	0	1	0	1
1	0	1		1	0	0	1	0
1	1	0		0	1	0	0	1
1	1	1		1	0	1	0	0

Circuits Logiques
(2) Circuits Séquentiels

Circuits séquentiels

- ◆ Circuits combinatoires
 - ◆ Les sorties ne dépendent que des valeurs des entrées
- ◆ Circuits séquentiels
 - ◆ Ajout des notions d'état et de mémoire
 - ◆ Ajout de la notion de temps (horloge)

Circuits séquentiels

- ◆ Les valeurs de sorties du circuit dépendent
 - ◆ Des valeurs en entrée
 - ◆ De valeurs calculées précédemment
 - ◆ De l'état dans lequel on se trouve
- ◆ Théories utilisées pour étudier/spécifier les différents types de circuits
 - ◆ Circuits combinatoires : algèbre de Boole
 - ◆ Circuits séquentiels : théorie des automates finis

Automate fini

- ◆ Un automate fini possède un nombre fini d'états
- ◆ Il est caractérisé, pour un couple d'instant $(t, t+1)$, par
 - ◆ Sa réponse S
 - ◆ Son entrée E
 - ◆ Son état Q

Automate fini

- ◆ 3 façons de déterminer un automate
 - ◆ Fonctions de transfert
 - ◆ $S(t+1) = f(Q(t), E(t))$: sortie à $t+1$ dépend des entrées et états à t
 - ◆ $Q(t+1) = g (Q(t), E(t))$: état à $t+1$ dépend des entrées et états à t
 - ◆ Tables de transition : tables des fonctions f et g
 - ◆ Valeurs de $Q(t+1)$ et $S(t+1)$ pour chaque combinaison de valeurs de $E(t)$ et $Q(t)$
 - ◆ Diagrammes d'états ou de transitions

Exemple d'automate fini

- ◆ Mémoire binaire : stocke 1 bit
- ◆ 1 entrée
 - ◆ Si 0, on mémorise la valeur 0
 - ◆ Si 1, on mémorise la valeur 1
- ◆ 2 états : valeur 0 ou valeur 1
- ◆ Principe
 - ◆ A t on est dans un état X , à $t+1$, on passe dans un état Y selon la valeur de l'entrée à t
 - ◆ La sortie S à $t+1$ prend la valeur de l'état à t

Exemple d'automate fini

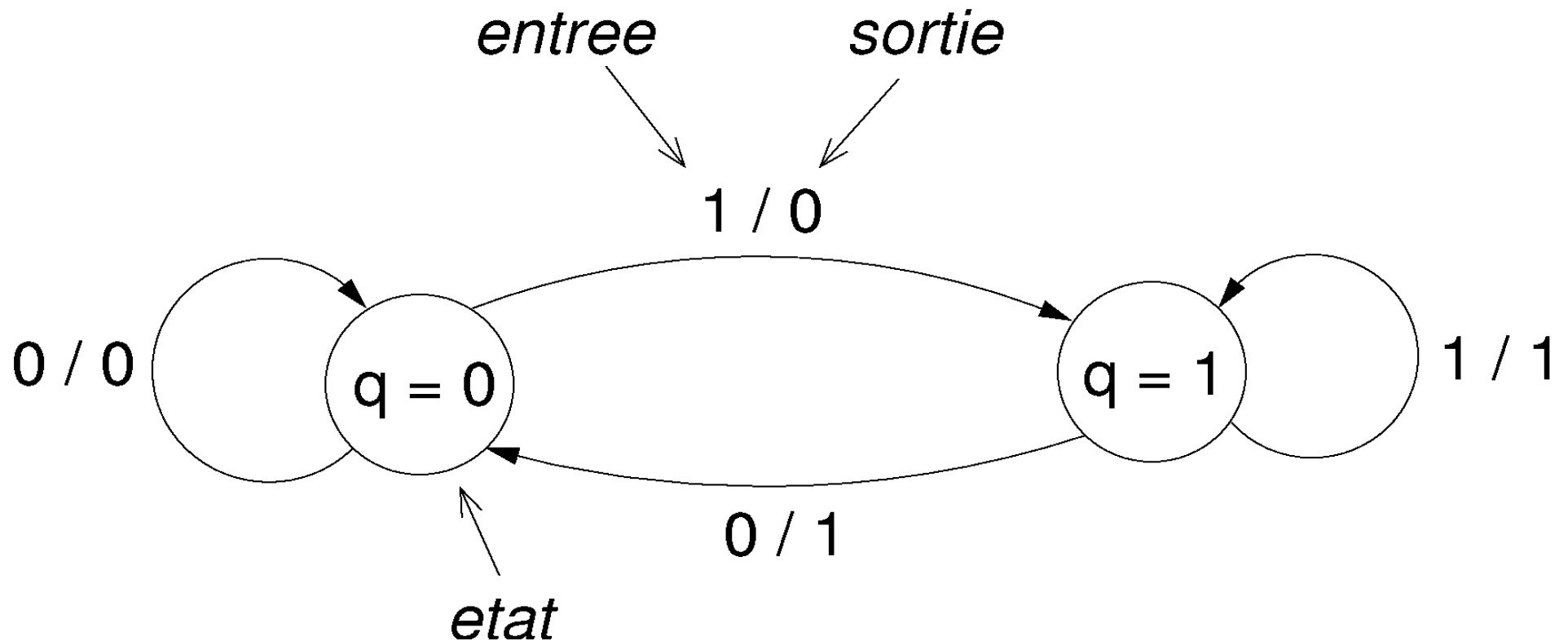
- ◆ Fonctions de transfert
 - ◆ $S(t+1) = Q(t)$: la sortie à $t + 1$ est égale à l'état à t
 - ◆ $Q(t+1) = E(t)$: l'état à $t + 1$ est égal à l'entrée passée à l'état t
- ◆ Tables de transitions : valeurs de $Q(t+1)$ et $S(t+1)$ en fonction de $E(t)$ et $Q(t)$

	$\backslash E(t)$		0	1
$Q(t)$		+ - - - -		
	0		0	0
	1		1	1
			$S(t+1)$	

	$\backslash E(t)$		0	1
$Q(t)$		+ - - - -		
	0		0	1
	1		0	1
			$Q(t+1)$	

Exemple d'automate fini

- ◆ Diagramme d'états/transitions



Caractéristiques électriques et temporelles

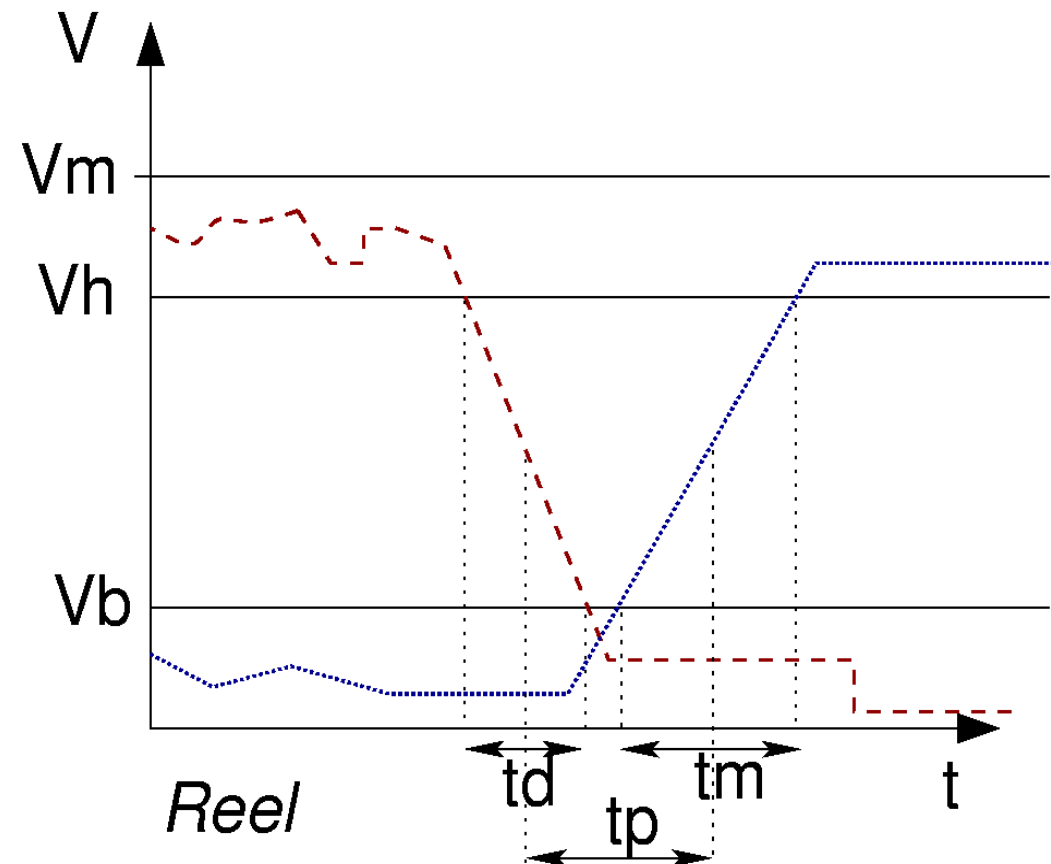
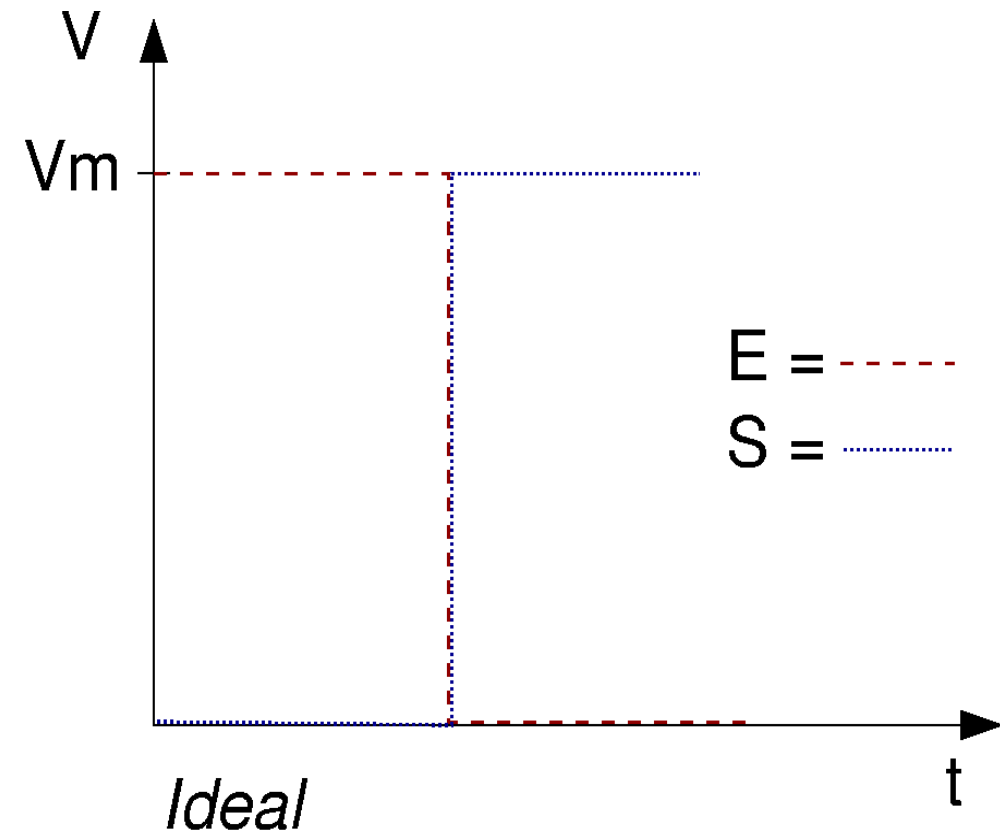
- ◆ D'un point de vue électrique et idéal
 - ◆ Un 0 correspond à 0 volt (parfois à $-V_m$ volts)
 - ◆ Un 1 correspond à $+V_m$ volts
- ◆ En pratique
 - ◆ 2 niveaux : niveau haut V_h et niveau bas V_b
 $0 < V_b < V_h < V_m$
 - ◆ Pour un voltage v
 - ◆ $0 < v < V_b$: représente un 0
 - ◆ $V_h < v < V_m$: représente un 1
 - ◆ $V_b < v < V_h$: indéfini

Caractéristiques électriques et temporelles

- ◆ Changement de valeur (0 -> 1 ou 1 -> 0)
 - ◆ Idéalement : instantané
 - ◆ En pratique : prend un certain temps, délai de montée (0 -> 1) et de descente (1 -> 0)
- ◆ Passage d'un signal au travers d'une porte
 - ◆ Idéalement : instantané
 - ◆ En pratique : prend un certain temps
 - ◆ Délai de propagation

Délais

- ◆ Exemple : traversée d'une porte NON
- ◆ $E = 1$ puis passe à 0, $S = \bar{E}$



Délais

◆ Voltages

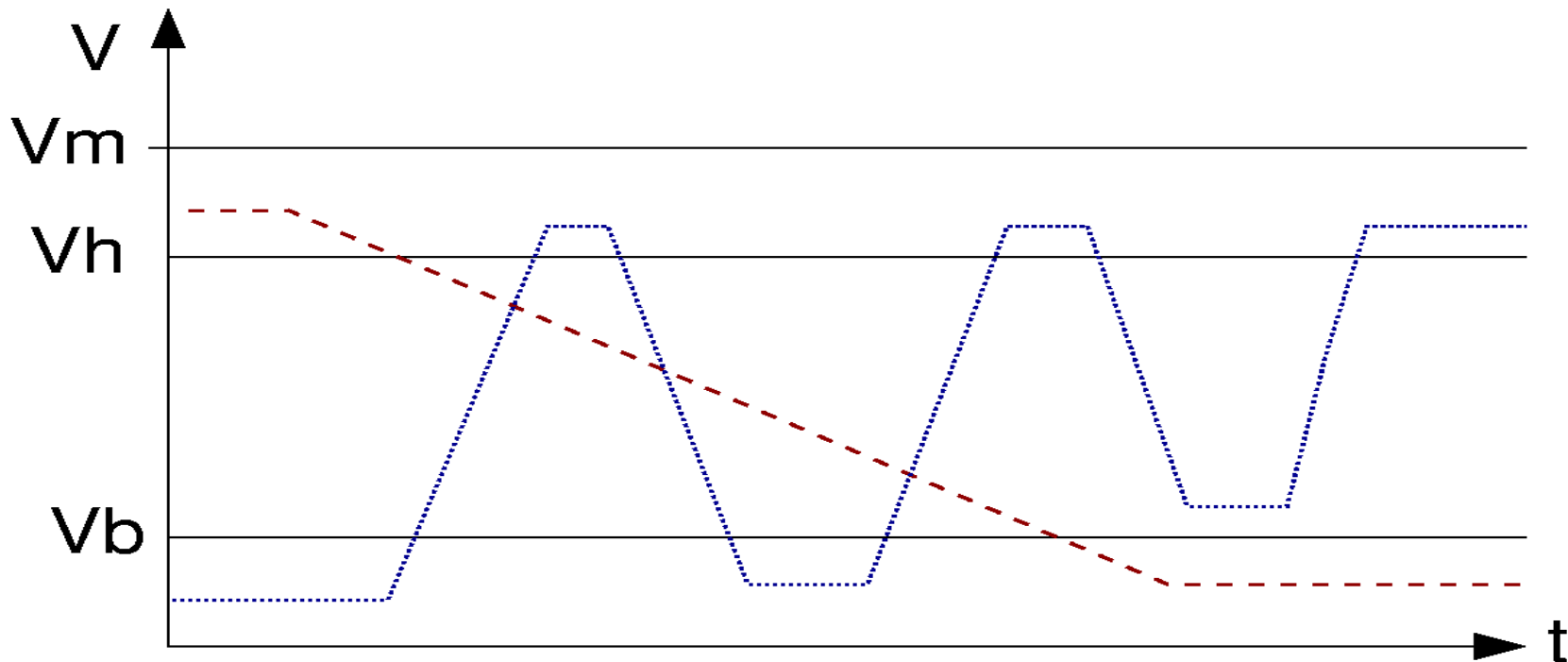
- ◆ V_m : voltage correspondant au 1 idéal
- ◆ V_h : voltage haut, minimum pour avoir un 1
- ◆ V_b : voltage bas, maximum pour avoir un 0

◆ Délais

- ◆ t_d : temps de descente, passage de 0 à 1 (voltage de V_h à V_b)
- ◆ t_m : temps de montée, passage de 0 à 1 (voltage de V_b à V_h)
- ◆ t_p : temps de propagation dans la porte (pris à la moyenne de V_h et V_b)

Délais

- ◆ Si temps de montée ou de descente trop long
 - ◆ On reste relativement longtemps entre V_b et V_h : état indéfini
 - ◆ La sortie oscille alors entre 0 et 1



Contraintes temporelles des circuits

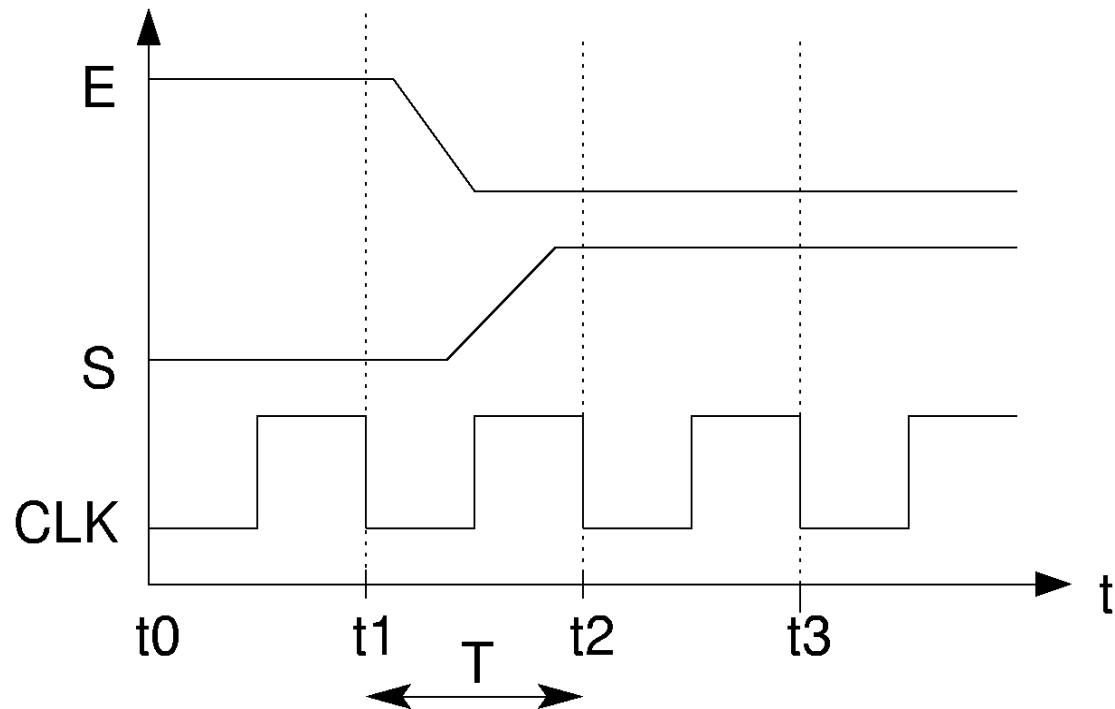
- ◆ Un circuit est formé de plusieurs portes
- ◆ Chemin critique : chemin le « plus long » pour la propagation des signaux à travers le circuit
- ◆ Détermine le temps total de propagation des signaux à travers tout le circuit
 - ◆ Temps minimal à attendre pour avoir une sortie valide
- ◆ Intuitivement : chemin passant par le plus grand nombre de portes
 - ◆ Mais dépend aussi du temps de propagation de chaque type de porte

Horloge

- ◆ A cause de tous les délais (montée, descente, propagation)
- ◆ Un signal n'est pas dans un état valide en permanence
- ◆ Idée : on ne lit ses valeurs qu'à des instants précis et à des intervalles réguliers
 - ◆ Instants donnés par une horloge
- ◆ Horloge
 - ◆ Indispensable aussi dans le cas des circuits séquentiels synchrones pour savoir quand on passe de t à $t + 1$

Horloge

- ◆ Horloge
 - ◆ Signal périodique
 - ◆ 1 demi-période à 0, l'autre à 1
 - ◆ Début d'une nouvelle période : instant t_i
- ◆ Pour exemple précédent du NON
 - ◆ Instant t_1 : $E = 1$, $S = 0$
 - ◆ Instant t_2 : $E = 0$, $S = 1$
 - ◆ CLK = Clock = signal d'horloge

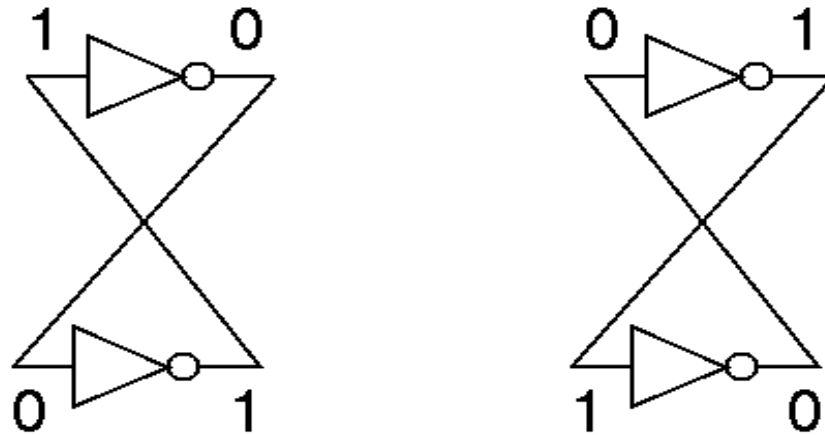


Mémoire sur 1 bit

- ◆ Circuit séquentiel basique : mémoire sur 1 bit
 - ◆ Possède une variable binaire codée sur 1 bit
 - ◆ Sa valeur est conservée ou modifiable dans le temps
- ◆ Mémoires 1 bit peuvent être réalisées avec des bascules

Bistable et bascule

- ◆ Bistable : 2 états stables dans le temps
- ◆ Principe général d'une bistable : 2 portes non (inverseurs) en opposition



- ◆ Bascule : composant qui met en oeuvre une bistable
- ◆ Possibilité de passer d'un état à l'autre, de changer l'état mémorisé
- ◆ Plusieurs façons de gérer et changer l'état
 - ◆ Plusieurs types de bascules : RS, D, JK ...

Bascule RS

- ◆ Entrée/sorties
 - ◆ 2 entrées : R et S
 - ◆ 1 sortie Q qui correspond à l'état stocké
- ◆ Principe : la valeur de Q à $t+1$ dépend de R, S et de la valeur de Q à t
 - ◆ R = reset : remise à 0 de Q
 - ◆ S = set : mise à 1 de Q
 - ◆ S=1 et R=0 : Q mis à 1
 - ◆ S=0 et R=1 : Q mis à 0
 - ◆ S=0 et R=0 : Q garde sa valeur, maintien
 - ◆ S=1 et R=1 : Q indéterminé

Bascule RS

- ◆ Table de vérité/transition du circuit et son tableau de Karnaugh

Q	R	S	Q ⁺
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	X
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	X

		RS			
		00	01	11	10
Q	0	0	1	X	0
	1	1	1	X	0

$$Q^+ = S + Q\bar{R}$$

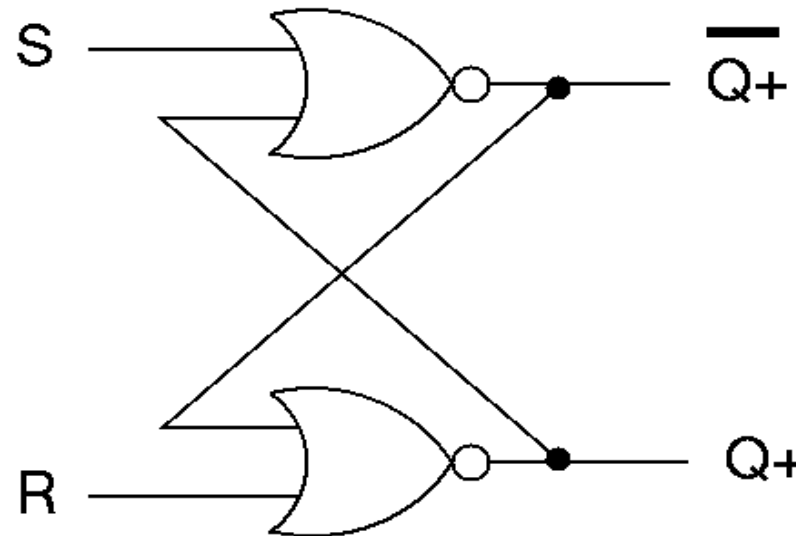
- ◆ Q⁺ : valeur de Q à l'instant suivant
- ◆ Dans tableau : X = valeur non connue mais sans influence, on peut regrouper dans les blocs à 1

Bascule RS

◆ Tables de vérité/transition dérivées

R	S		Q+	Q	->	Q+		R		S
0	0		Q	0		0		X		0
0	1		1	1		1		0		X
1	0		0	0		1		0		1
1	1		X	1		0		1		0

◆ Logigramme de la bascule RS, avec des portes NOR



3 types de bascules

◆ Bascule asynchrone

- ◆ Sorties recalculées à chaque changement des valeurs en entrée
 - ◆ Plus précisément : recalculées « en permanence »
- ◆ Exemple : la bascule RS précédente

◆ Bascule synchrone

- ◆ Sorties recalculées en fonction d'un signal d'horloge en entrée
 - ◆ Nécessite une entrée de contrôle pour ce signal d'horloge
- ◆ 2 modes
 - ◆ Sur niveau (dit « latch »)
 - ◆ Recalcule les sorties et l'état quand le signal d'horloge est à 1
 - ◆ Sur front (dit « flip-flop »)
 - ◆ Recalcule au moment où le signal d'horloge change de valeur
 - ◆ Front montant : passage de 0 à 1
 - ◆ Front descendant : passage de 1 à 0
- ◆ Note : on utilise en général un signal d'horloge sur l'entrée de contrôle, mais n'importe quel signal peut être utilisé en pratique

3 types de bascules

- ◆ Détermination du temps suivant (passage de t à $t + 1$) selon le type de bascule
 - ◆ Asynchrone
 - ◆ Quand les entrées changent et la sortie est recalculée
 - ◆ Synchrones sur niveau
 - ◆ Quand le niveau (1 en général, mais 0 aussi) est atteint
 - ◆ Synchrones sur front
 - ◆ Au moment du passage de 0 à 1 ou de 1 à 0 selon le type de front utilisé par la bascule
- ◆ Dans tous les cas
 - ◆ Quand on recalcule, les valeurs prises pour l'état et les entrées sont celles *juste* avant de passer à $t + 1$, quand on est encore à t

Bascule D latch

- ◆ D = delay
- ◆ 2 entrées
 - ◆ D : la valeur en entrée
 - ◆ C : entrée de contrôle, généralement un signal d'horloge
- ◆ 1 sortie Q
 - ◆ $Q^+ = D$ si $C = 1$
 - ◆ $Q^+ = Q$ si $C = 0$
- ◆ Si C est un signal d'horloge
 - ◆ Retarde le signal en entrée D d'une période d'horloge

Bascule D latch

◆ Table de vérité de la bascule D

D	C	Q		Q+
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		1
1	1	1		1

$$Q+ = DC + Q\bar{C}$$

Versions condensées :

(C=1)	D		Q+
	0		0
	1		1

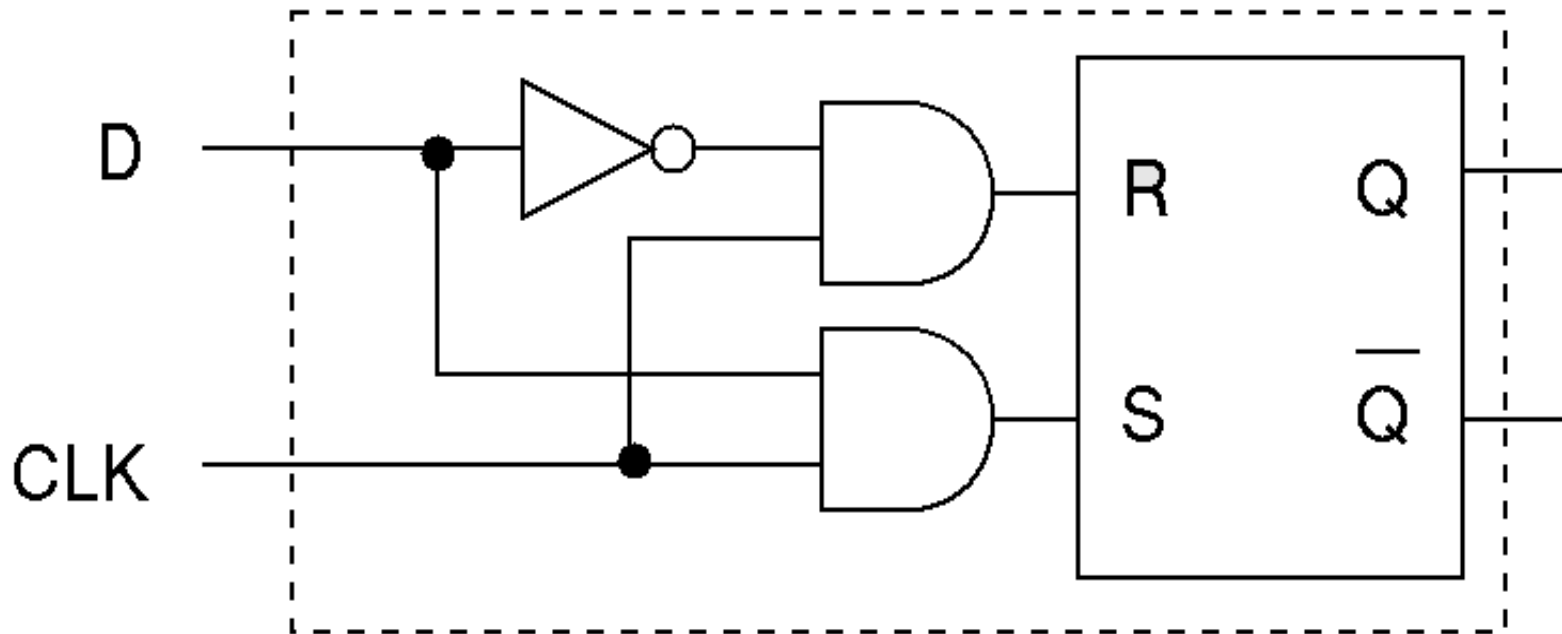
C		Q+
0		Q
1		D

Bascule D latch

- ◆ Réalisation d'une bascule D à partir d'une bascule RS
 - ◆ $S = CD$ et $R = C\bar{D}$
- ◆ Pour S
 - ◆ Pour bascule D, $Q_+ = 1$ si $C = 1$ et $D = 1$
 - ◆ Pour bascule RS, $Q_+ = 1$ si $S = 1$ et $R = 0$
- ◆ Pour R
 - ◆ Pour bascule D, $Q_+ = 0$ si $C = 1$ et $D = 0$
 - ◆ Pour bascule RS, $Q_+ = 0$ si $S = 0$ et $R = 1$
- ◆ Avec $S = CD$ et $R = C\bar{D}$, tout est cohérent
- ◆ Et le maintien $Q_+ = Q$ de la bascule D lorsque $C=0$ est cohérent avec le maintien de la RS : $R=S=0$

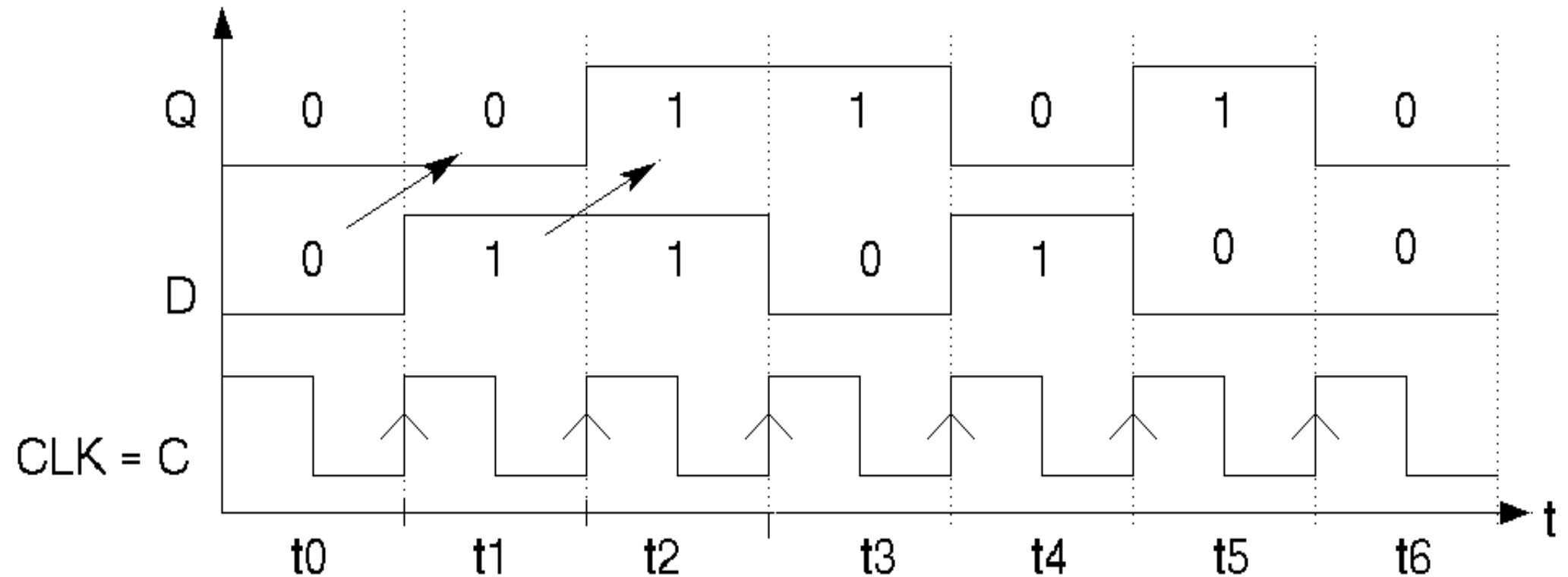
Bascule D latch

- ◆ Logigramme de la bascule D latch



Bascule D flip-flop

- ◆ Bascule D flip-flop
 - ◆ Variante de la D latch
 - ◆ Passage à $t + 1$ quand front montant de l'horloge
 - ◆ En pratique, la D latch fonctionne de la même façon
- ◆ Exemple de chronogramme avec $D = 0110100$ et $Q(0) = 0$



Bascule JK asynchrone

- ◆ JK = « rien de particulier », variante de RS
- ◆ Comme pour RS mais ajoute le cas de R=S=1
- ◆ Si J = K = 1 alors $Q_+ = \overline{Q}$

J	K	Q	Q ⁺
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$Q_+ = J\overline{Q} + \overline{K}Q$$

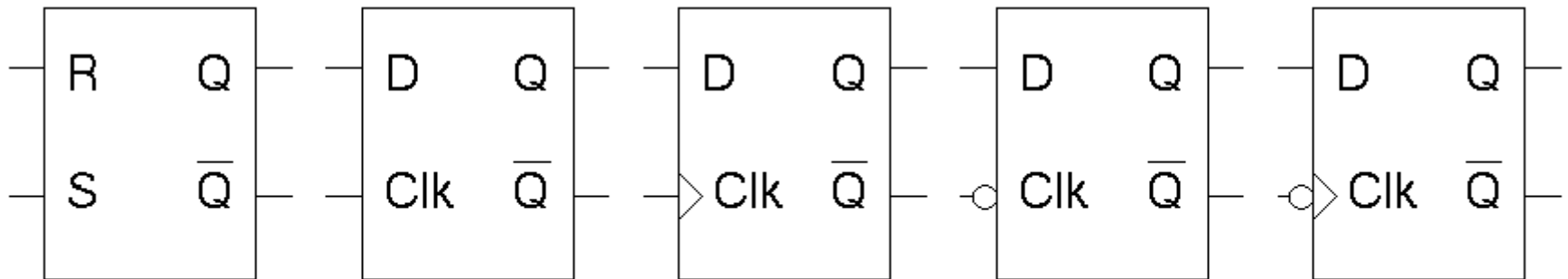
		JK						
Q	\	00	01	11	10	J	K	Q ⁺
		0	0	0	1	1	0	0
1	1	0	0	1	1	0	1	0
1	1	1	0	0	1	1	1	\overline{Q}

Représentation des différents types de bascule

- ◆ Bascule asynchrone
 - ◆ Pas d'entrée CLK ou de contrôle
- ◆ Bascule synchrone
 - ◆ Entrée CLK
 - ◆ Synchrone sur front : rajoute un triangle sur l'entrée de contrôle
 - ◆ Si « rond inverseur »
 - ◆ Bascule latch
 - ◆ Modifie la sortie quand valeur d'horloge à 0 (1 par défaut)
 - ◆ Bascule flip-flop
 - ◆ Actif sur front descendant (montant par défaut)

Représentation des différents types de bascule

- ◆ De gauche à droite
 - ◆ Bascule RS asynchrone
 - ◆ Bascule D latch sur niveau 1
 - ◆ Bascule D flip-flop sur front montant
 - ◆ Bascule D latch sur niveau 0
 - ◆ Bascule D flip-flop sur front descendant

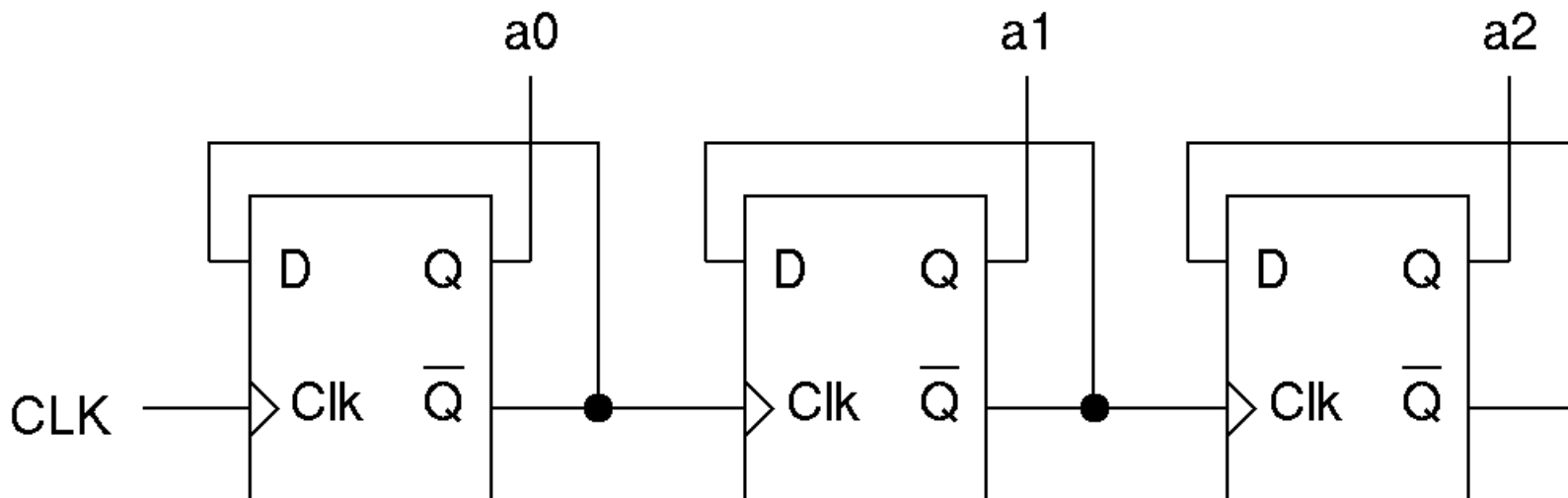


Utilisation des bascules

- ◆ Utilisation des bascules pour créer des circuits ayant un état
 - ◆ Compteurs
 - ◆ Registres
 - ◆ Mémorisation d'un mot mémoire, décalage vers la droite/gauche du mot ...
 - ◆ Mémoires (voir le cours sur les mémoires)
- ◆ Exemple : compteur cyclique sur 3 bits
 - ◆ Valeur en décimal sur 3 bits
 - ◆ Incrémentation de +1 à chaque période d'horloge
 - ◆ Repasse à 0 après 7

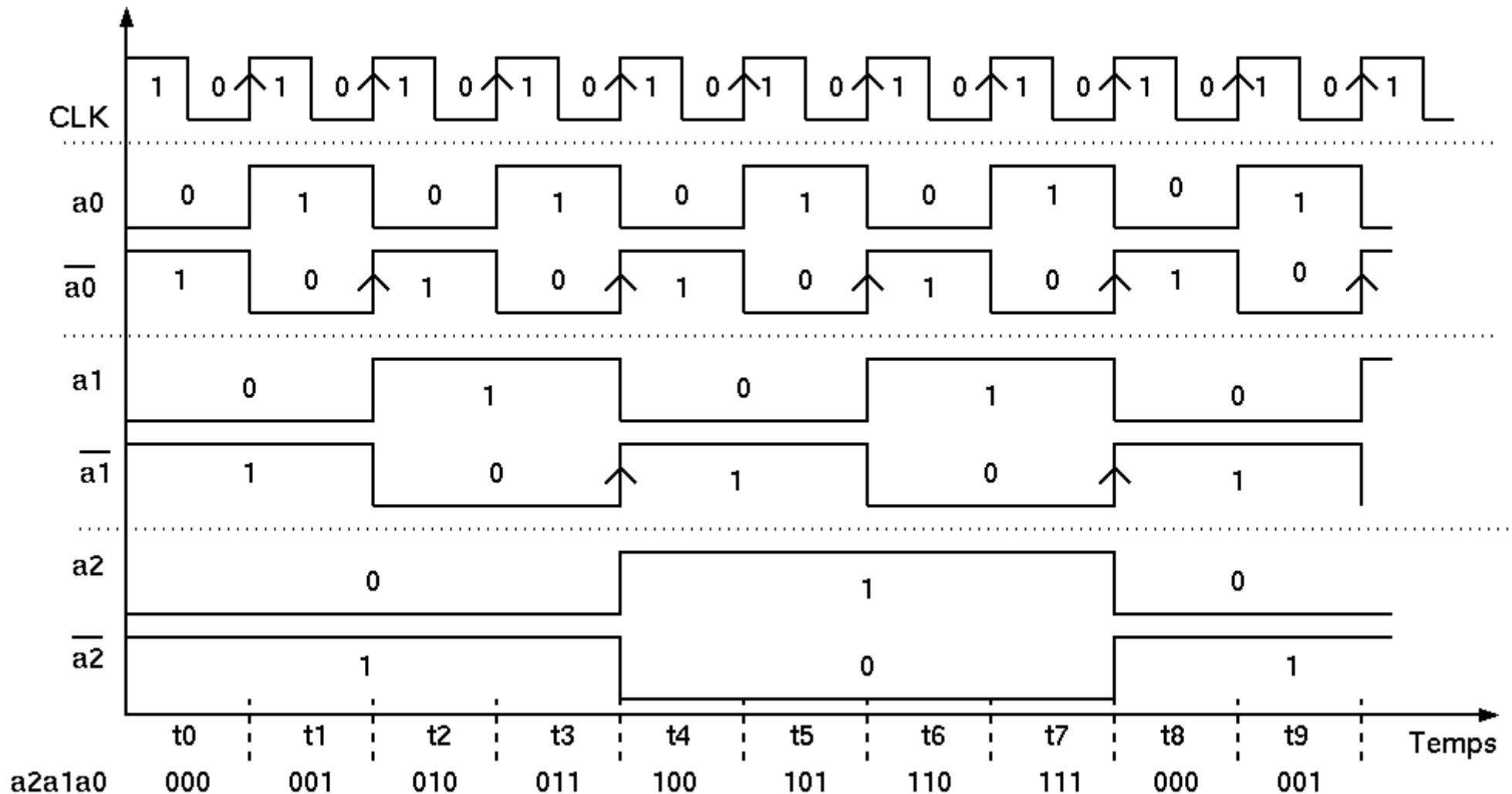
Compteur 3 bits

- ◆ Utilisation de 3 bascules D
 - ◆ Principe
 - ◆ Chaque bascule prend en entrée D un signal d'horloge
 - ◆ Fournit en sortie un signal d'horloge de fréquence divisée par 2
 - ◆ En mettant en série les 3 bascules
 - ◆ 3 signaux d'horloge à 3 fréquences différentes
 - ◆ Représente les combinaisons de bits pour les valeurs de 0 à 7



Compteur 3 bits

- ◆ Chronogramme du compteur 3 bits
- ◆ Version idéale, ne prend pas en compte les temps de propagation à travers les bascules (voir TD/TP)

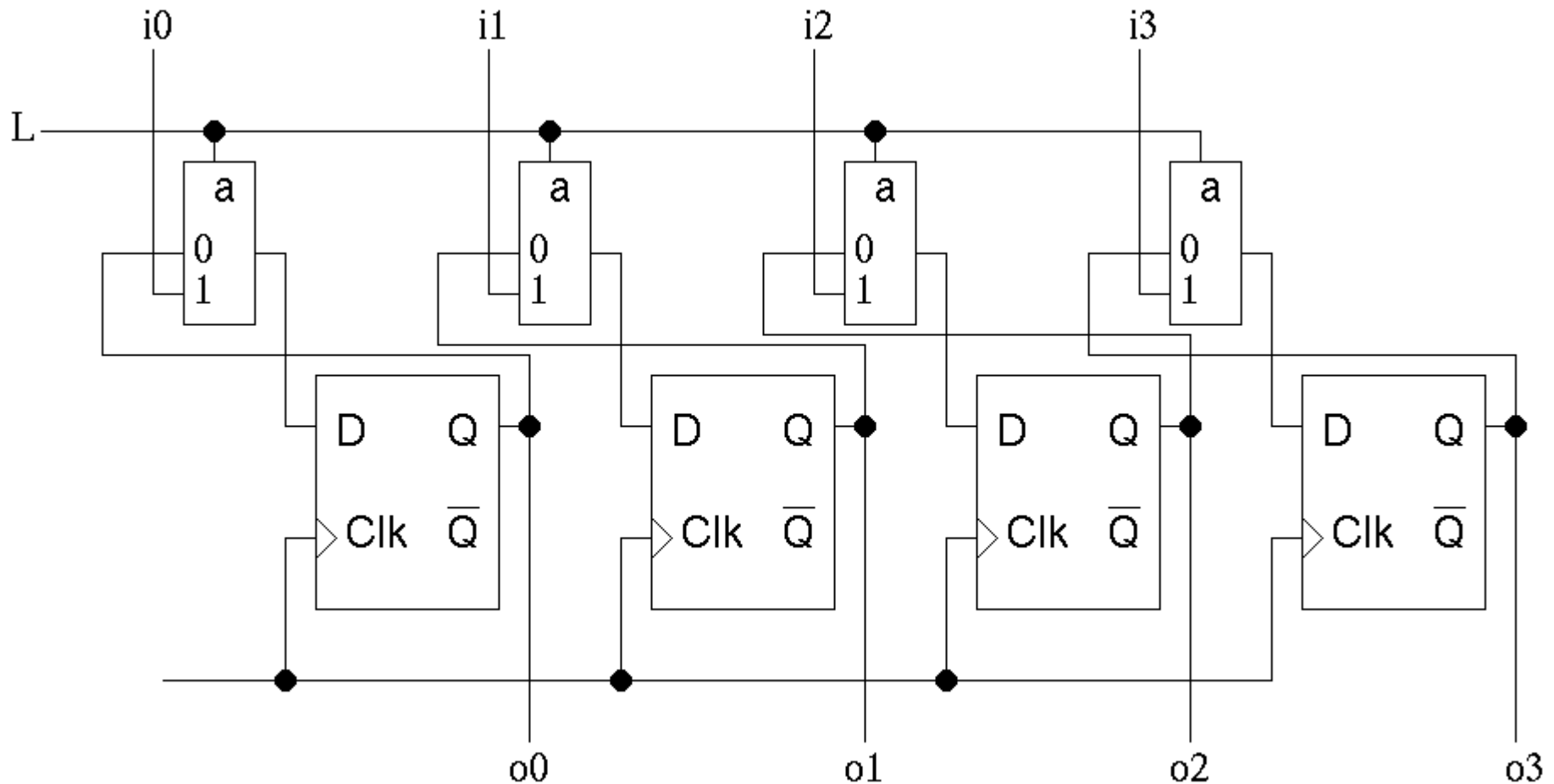


Registre

- ◆ Registre : mot mémoire de X bits
 - ◆ Composant utilisé en interne dans un processeur pour stocker des valeurs lors de l'exécution d'un programme par ce processeur
- ◆ Exemple : registre 4 bits
 - ◆ 4 bascules D stockent les 4 bits
 - ◆ 4 bits I_x en entrée pour écrire le mot
 - ◆ 4 bits O_x en sortie pour récupérer la valeur du mot
 - ◆ Une entrée L (pour « load ») précise si on conserve la valeur du mot stocké (valeur 0) ou écrit le mot avec les 4 bits en entrée (valeur 1)

Registre 4 bits

- ◆ Selon la valeur de L , à chaque front montant de l'horloge, les sorties des multiplexeurs prennent les valeurs i_x ou reprennent les valeurs déjà stockées par les bascules



Circuits synchrone et asynchrone

- ◆ Circuit synchrone
 - ◆ Tous les éléments/composants du circuit devant être synchronisés le sont avec le même signal d'horloge
- ◆ Circuit asynchrone
 - ◆ Tous les éléments/composants du circuit devant être synchronisés ne le sont pas avec le même signal d'horloge

Circuits synchrone et asynchrone

- ◆ Exemple du compteur 3 bits
 - ◆ Circuit est asynchrone car plusieurs signaux d'horloges différents utilisés par les bascules
 - ◆ Chaque bascule calcule donc sa sortie à des moments différents
 - ◆ Même si certains sont communs à plusieurs bascules
 - ◆ Les bascules D utilisées sont des bascules synchrones flip-flop mais le circuit lui est bien asynchrone
- ◆ Exemple du registre 4 bits
 - ◆ Synchrone car les 4 bascules D flip-flop utilisent toutes les 4 le même signal d'horloge