

# Conception des systèmes numériques

## 3. Logique Combinatoire

Principaux circuits arithmétiques

# Circuits combinatoires & séquentiels

- **Circuits Combinatoires**

Les fonctions sont réalisées à partir d'opérateurs  
(AND,OR,NOT...).

Les sorties de ces fonctions dépendent uniquement des entrées

Tout changement d'une des variables d'entrées est propagé à travers les différentes portes de la fonction

Combinatoire :

$S = f(a,b,c,..,n)$  → Boucle ouverte

- **Circuits séquentiels**

Les sorties dépendent des entrées, de la sortie à l'instant précédent et de l'ordre dans lequel elles sont appliquées

Séquentiel :       $S = f(a,b,c,..,n,S)$  → Système contre réactionné

# Circuits combinatoires & séquentiels

- **Logique combinatoire : modélisation par algèbre de Boole**
  - Comparateur
  - Décodeur
  - Multiplexeur
  - UAL : Unité Arithmétique et Logique  
(Additionneur, soustracteur, multiplicateur...)
- **Logique séquentielle : modélisation dynamique**
  - Bascules
  - Horloges
  - Compteurs
  - Registres...

# Comparateurs

## Fonctions Combinatoires Complexes

- Egalité 2 mots de 2 bits :  $a = a_1, a_0$  et  $b = b_1, b_0$

$\begin{matrix} \bar{a}_1 \\ a_0 \end{matrix}$	0	0	1	1
$\begin{matrix} b_1 \\ b_0 \end{matrix}$	0	1	1	0
00	1			
01		1		
11			1	
10				1

$$S = (\overline{a_1 \oplus b_1}) \cdot (\overline{a_0 \oplus b_0})$$

- Egalité 2 mots de  $n$  bits :  $a = a_{n-1}, \dots, a_0$  et  $b = b_{n-1}, \dots, b_0$

$$S = (\overline{a_{n-1} \oplus b_{n-1}}) \cdot (\overline{a_{n-2} \oplus b_{n-2}}) \cdot (\dots) \cdot (\overline{a_1 \oplus b_1}) \cdot (\overline{a_0 \oplus b_0})$$

# Comparateurs

- Egalité 2 mots de n bits : description VHDL - approche fonctionnelle

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

entity egalite_nb_fonc is
    generic(N : integer:= 4);
    port(
        a,b           : in      std_logic_vector(N-1 downto 0);
        s             : out      std_logic);
end egalite_nb_fonc;
architecture comportement of egalite_nb_fonc is
begin
process(a,b) is
    begin
        if (a=b) then
            s<='1';
        else
            s<='0';
        end if;
    end process;
end comportement;
```

# Comparateurs

- Egalité 2 mots de n bits : description VHDL  
architecture avec génération

$$S = (\overline{a_{n-1} \oplus b_{n-1}}) \cdot (\overline{a_{n-2} \oplus b_{n-2}}) \cdot (\dots) \cdot (\overline{a_1 \oplus b_1}) \cdot (\overline{a_0 \oplus b_0})$$

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

entite egalite_nb_struct is
    GENERIC(N : integer:= 4);
    PORT(a,b      : in std_logic_vector(N-1 downto 0);
          s : out std_logic);
end egalite_nb_struct;

architecture comportement OF egalite_nb_struct is
signal temp : std_logic_vector(n-1 downto 0);
begin
    temp(0)<='1';
    boucle:
        FOR i in 1 to N-1 GENERATE
            temp(i)<=not(a(i) xor b(i)) and not(a(i-1) xor b(i-1)) and temp(i-1);
        end generate;
        s<=temp(n-1);
end comportement;
```

# Multiplexeurs

- Multiples informations = multiples variables → canal unique (fonction)
- Exemple:
  - Transmission de données numériques parallèles vers une transmission série (sortie d'un CAN parallèle-série)
  - Concentrateurs réseaux...
- Sélection aiguillage : commande codée
- Multiplexeur n variables en 1
- Partie Commande : n bits
- Partie Données :  $2^n = N$  entrées, 1 sortie

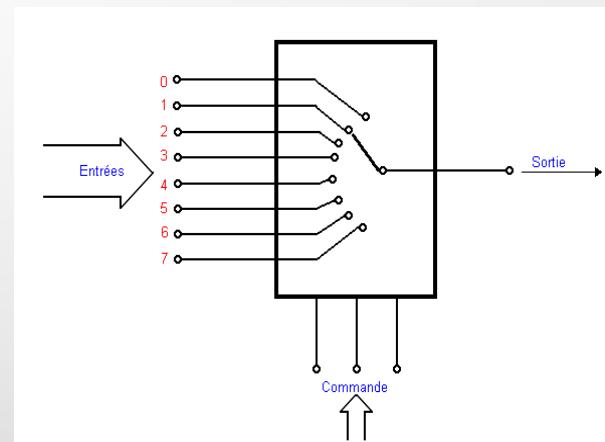
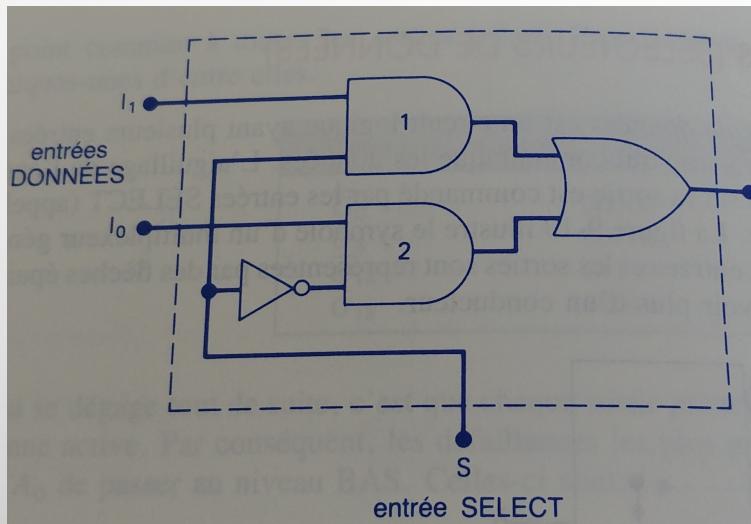


Fig. 25. - Schéma de principe d'un commutateur numérique (multiplexeur) à huit entrées (8 vers 1).

# Multiplexeurs

Exemple : Multiplexeurs 2 vers 1 - Table de vérité

- Partie Commande : 1 bits
- Partie Donnée :  $2^1 = 2$  entrées, 1 sortie



$$\text{Sortie} = \bar{S} \cdot I_0 + S \cdot I_1$$

# Multiplexeurs

## Fonctions Combinatoires Complexes

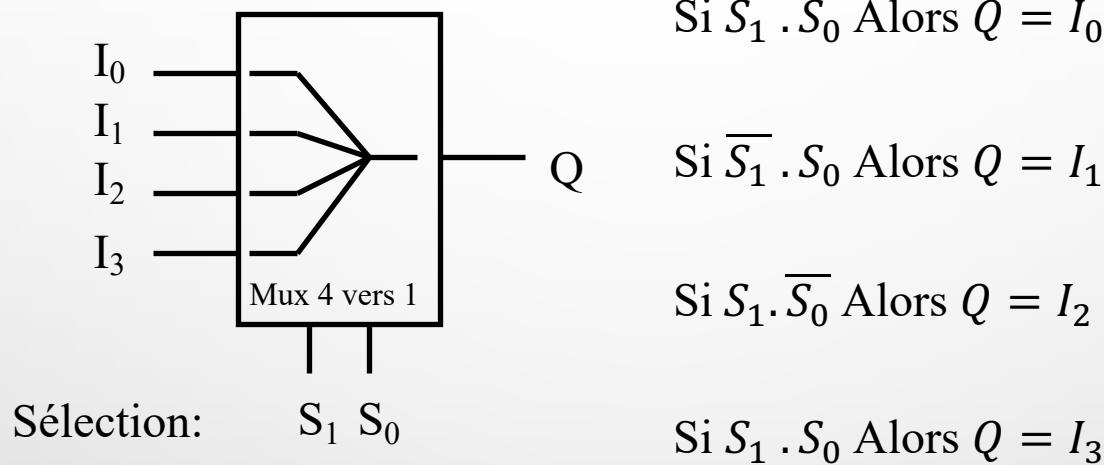
➤ Multiplexeurs 2 vers 1 – description VHDL

```
entity m2v1 is
port(      a,b,sel : in std_logic;
            s: out std_logic);
end entity m2v1;
```

```
architecture comportement of m2v1 is
Begin
Process (sel)
begin
s <= a when sel='0' else b;    -- s prend la valeur de a si sel = '0'
                                -- sinon s prend la valeur de b
end process;
end comportement;
```

# Multiplexeur

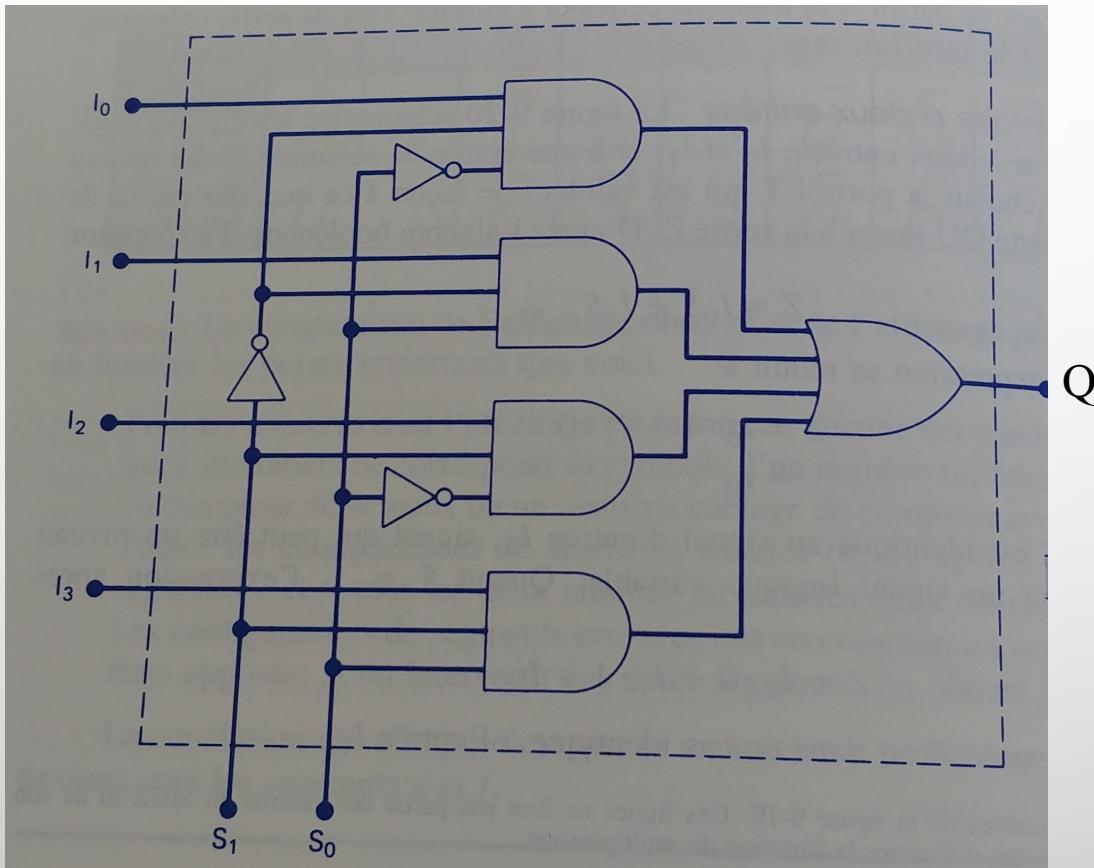
Sélection d'une voie parmi  $2^n$  par n bits de commande



$$\text{Alors } Q = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$

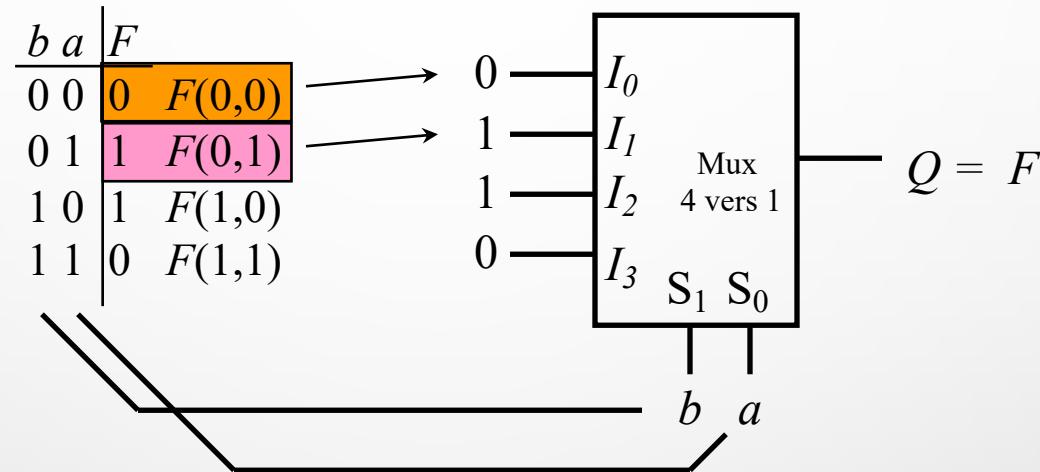
# Multiplexeur

$$Q = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$



# Multiplexeur et Fonctions

Utilisation de la première forme canonique

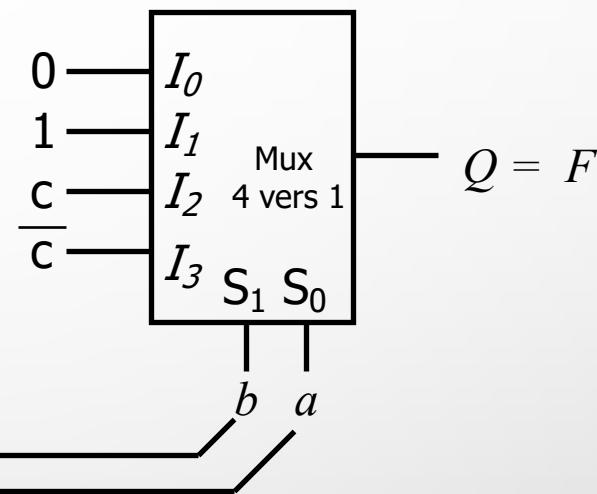


$$Q = \bar{b} \cdot \bar{a} \cdot I_0 + \bar{b} \cdot a \cdot I_1 + b \cdot \bar{a} \cdot I_2 + b \cdot a \cdot I_3$$

Toute fonction logique de  $n$  variables est réalisable avec un multiplexeur de  $2^n$  vers 1

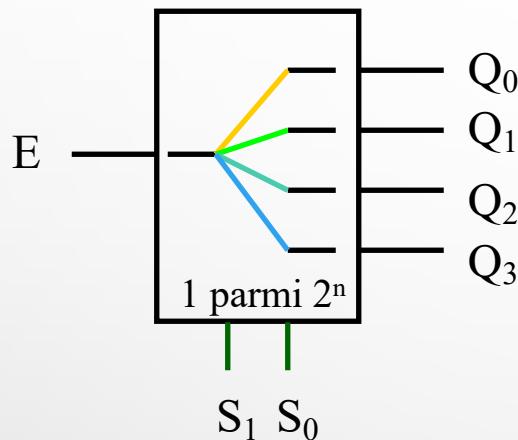
# Multiplexeur et fonctions

$b$	$a$	$c$	$F$
0	0	0	0 $(ba)_2 = 0$
0	0	1	0 $F = 0$
0	1	0	1 $(ba)_2 = 1$
0	1	1	1 $F = 1$
1	0	0	0 $(ba)_2 = 2$
1	0	1	1 $F = c$
1	1	0	1 $(ba)_2 = 3$
1	1	1	0 $F = c$



Toute fonction logique de  $n$  variables est réalisable avec un multiplexeur de  $2^{n-1}$  vers 1 et un inverseur

# Démultiplexeur



$$Q_0 = E \text{ si } (S_1 S_0)_2 = 0$$

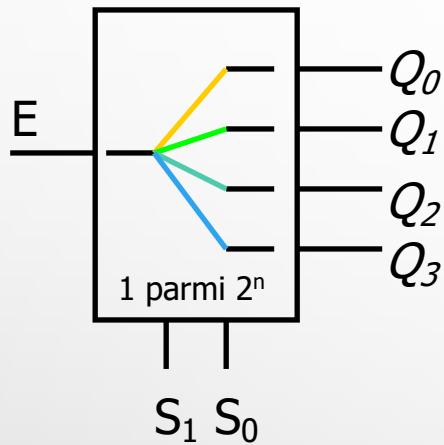
—  
E sinon

$$Q_1 = E \text{ si } (S_1 S_0)_2 = 1$$

—  
E sinon

Remarque :  $E$  peut ne pas être «disponible» (asynchrone)  
Sortie sélectionnée = 1 les autres 0  
ou Sortie sélectionnée = 0 les autres 1

# Démultiplexeur



$$Q_0 = E \cdot \overline{S_1} \cdot \overline{S_0} + \overline{E} \cdot \overline{\overline{S_1} \cdot \overline{S_0}}$$
$$= E \cdot \overline{S_1} \cdot \overline{S_0} + \overline{E} \cdot (S_1 + S_0)$$

$$Q_1 = E \cdot \overline{S_1} \cdot S_0 + \overline{E} \cdot \overline{\overline{S_1} \cdot S_0}$$
$$= E \cdot \overline{S_1} \cdot S_0 + \overline{E} \cdot (S_1 + \overline{S_0})$$

E=1

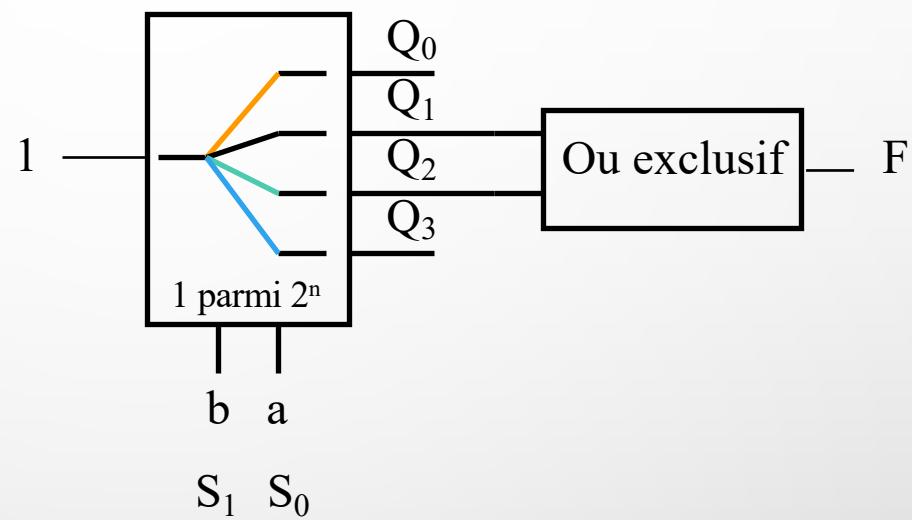
$$Q_0 = \overline{S_1} \cdot \overline{S_0}$$

$$Q_1 = \overline{S_1} \cdot S_0$$

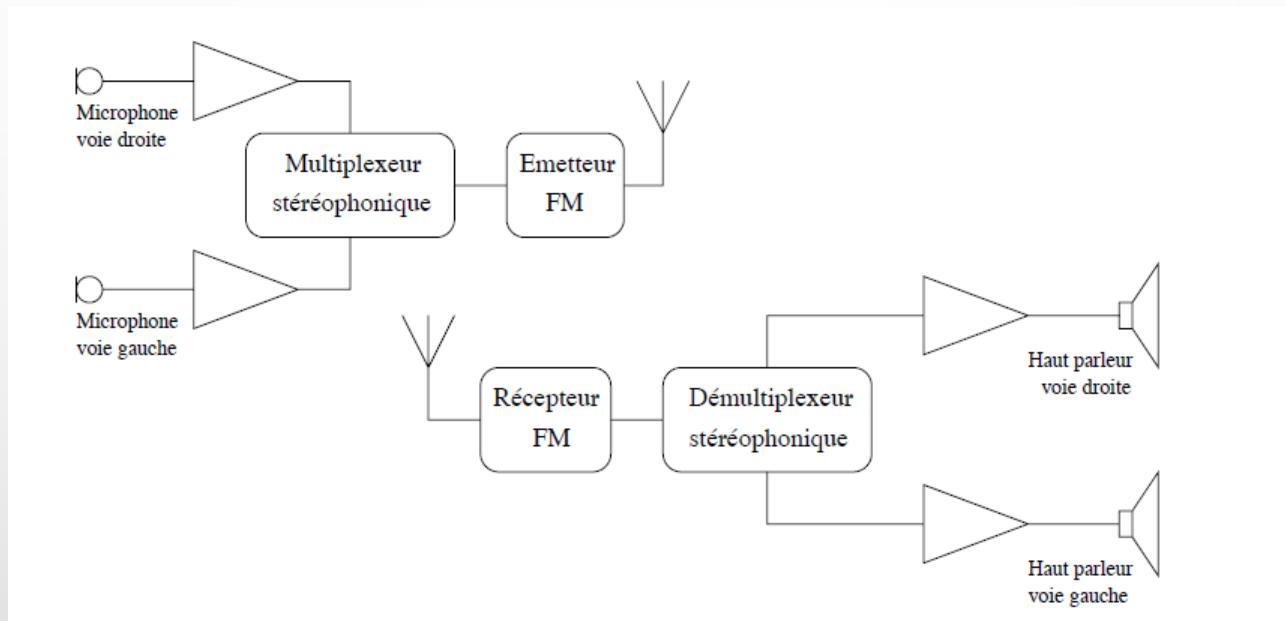
$$Q_i = (i)_2$$

# Demultiplexeur

b	a	F
0	0	0 F(0,0)
0	1	1 F(0,1)
1	0	1 F(1,0)
1	1	0 F(1,1)



# Multiplexeur-Démultiplexeur

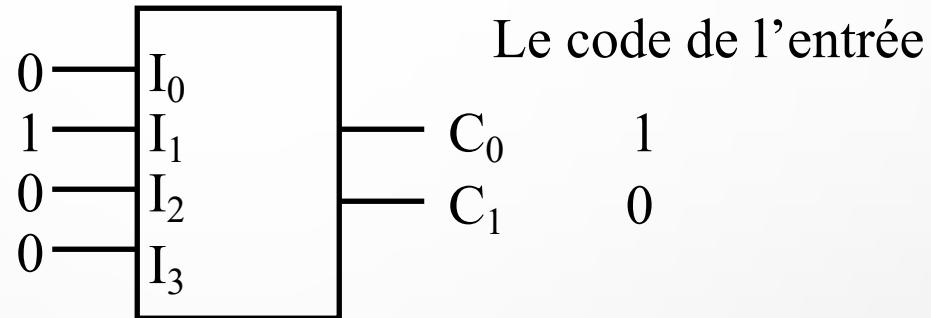


Transmission stéréophonique

# Codeur

Faire correspondre un mot code à un symbole

1 entrée parmi  $N=2^n$



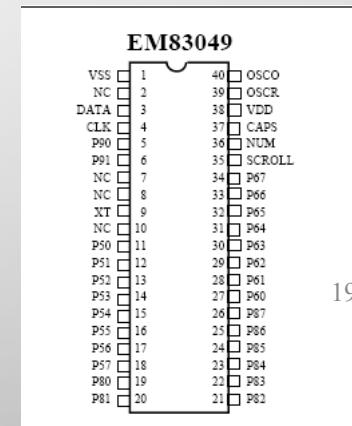
Exemples :

• Clavier → keyboard Scan code (Pour un clavier à 84 touches : 26 lettres minuscules, 26 lettres majuscules, 10 chiffres et 22 caractères divers), il faut donc 7 bits de sortie ( $2^7=128$ ) pour coder ces 84 touches du clavier

EM83049 microcontrôleur est dédié au « keyboard encoder »

Caractère → Code ASCII

Symbol LS3578 Motorola



## Exemple : Codeur binaire 8 vers 3 (8 entrées vers 3 sorties)

Ce codeur reçoit une information codée sur une de ses huit entrées et génère l'équivalent binaire sur les sorties  
Une seule entrée doit être active à la fois

Entrée activée(=1)	S2	S1	S0
E0	0	0	0
E1	0	0	1
E2	0	1	0
E3	0	1	1
E4	1	0	0
E5	1	0	1
E6	1	1	0
E7	1	1	1

$$S0 = E1 + E3 + E5 + E7$$

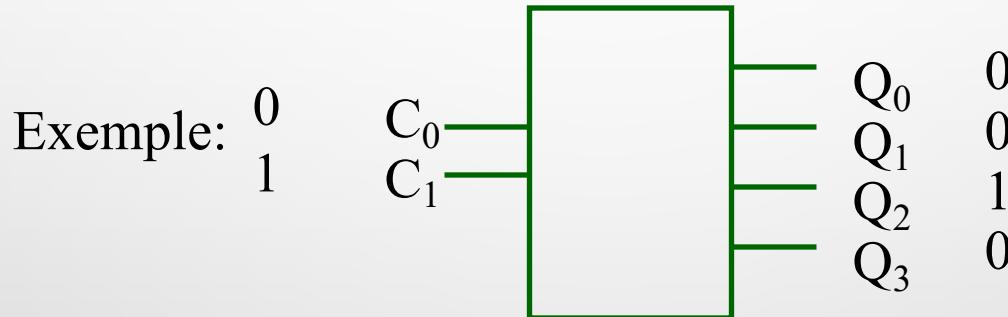
$$S1 = E2 + E3 + E6 + E7$$

$$S2 = E4 + E5 + E6 + E7$$

# Décodeur

Remarque : Multiplexeur  $\leftrightarrow$  Démultiplexeur  
Codeur  $\leftrightarrow$  Décodeur

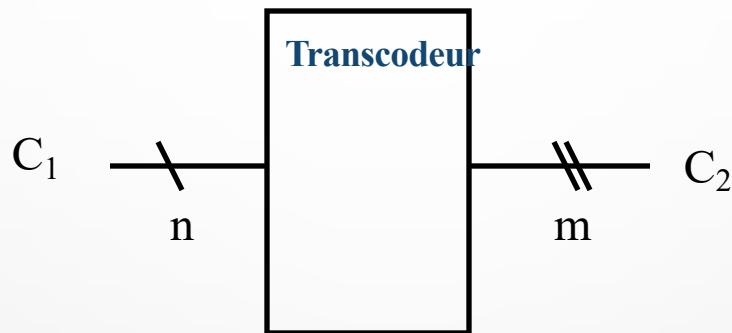
Décodeur = Démultiplexeur (à E fixe)



Exemple : adresses pixel / position effective pixel

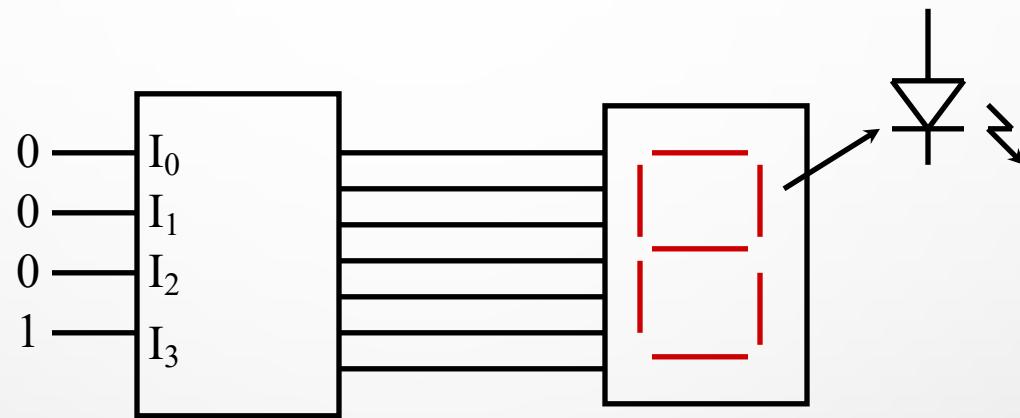
# Transcodeur = Codec

Passage d'un code quelconque  $C_1$  à un autre code  $C_2$



- Le transcodage permet d'adapter le format du média au support sur lequel il est transporté, stocké ou diffusé
- Les capacités de transport de l'information, en matière de bande passante pour la diffusion hertzienne et de débit pour les réseaux informatiques, sont décisives si le média doit être diffusé en streaming ou en broadcasting.
- TNT, MPEG2 → MPEG4, GIF → JPEG

# Transcodeur : exemple



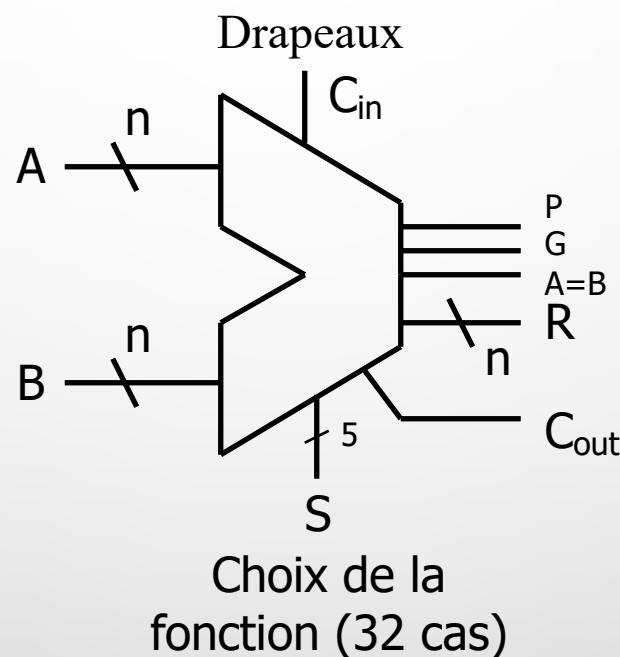
Code binaire 0 à 9

Configuration alimentation  
des diodes (ou LCD)

# Unité Arithmétique et logique

## UAL (ALU)

Remarque : architecture des machines



Sortie  
Exemple :

$$R = A + B$$
$$R = A + \overline{B}$$
$$R = A + B + 1$$

...

$$R = A \text{ ou } B$$
$$R = A \text{ nand } B$$

...

# Fonctions arithmétiques : Additionneur

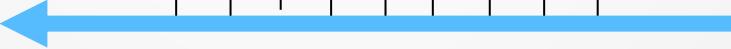
Conception « traditionnelle »

- Table de vérité
- Équations logiques
- Minimisation suivant un critère (coût, nombre de portes, rapidité...)
- Schéma, réalisation
- Problème : si A et B sont des mots exprimés sur 32 bits
- $2^{64}$  ( $\sim 10^{19}$ ) combinaisons possibles!!!

→ Méthode impraticable

# Fonctions arithmétiques : Additionneur

L'addition décimale

$$\begin{array}{r} 9276275382 \\ \text{Plus} \ 5873418752 \\ \hline 134 \end{array}$$


Le résultat s'obtient étage par étage

L'addition binaire: 1 bit

$$0 \text{ et } 0 = 0$$

$$0 \text{ et } 1 = 1$$

$$1 \text{ et } 0 = 1$$

$$1 \text{ et } 1 = 0 \text{ report (carry) } 1$$

# Fonctions arithmétiques :

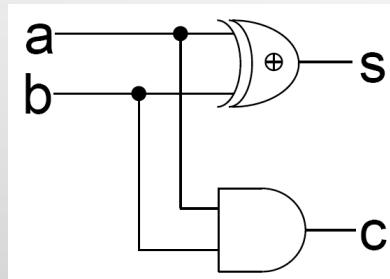
## Additionneur

- Réalisation d'un demi-additionneur

$$s = a \oplus b$$

$$r = c = a.b$$

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



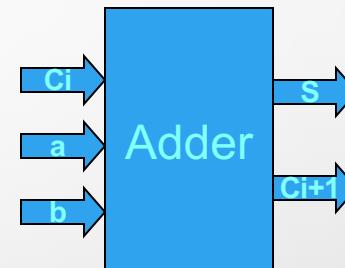
```
entity demi-add is
port(  a,b : in std_logic;
       s,c : out std_logic);
end entity demi-add;

architecture comportement of demi-add is
begin
  s<= a xor b;
  c<= a and b;
end architecture flot;
```

# Fonctions arithmétiques :

## Additionneur

- Circuit combinatoire : additionneur entier
  - ◆ 3 entrées:
    - ◆ 2 variables binaires  $a$  et  $b$
    - ◆ Retenue (carry) éventuelle  $C_{in}$
  - ◆ 2 sorties
    - ◆ Résultat (somme)  $S$
    - ◆ Retenue éventuelle  $C_{out}$



# Fonctions arithmétiques :

## Additionneur

Réalisation d'un additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées,  $a_i, b_i, c_i$  et deux de sortie  $s_i, c_{i+1}$

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$

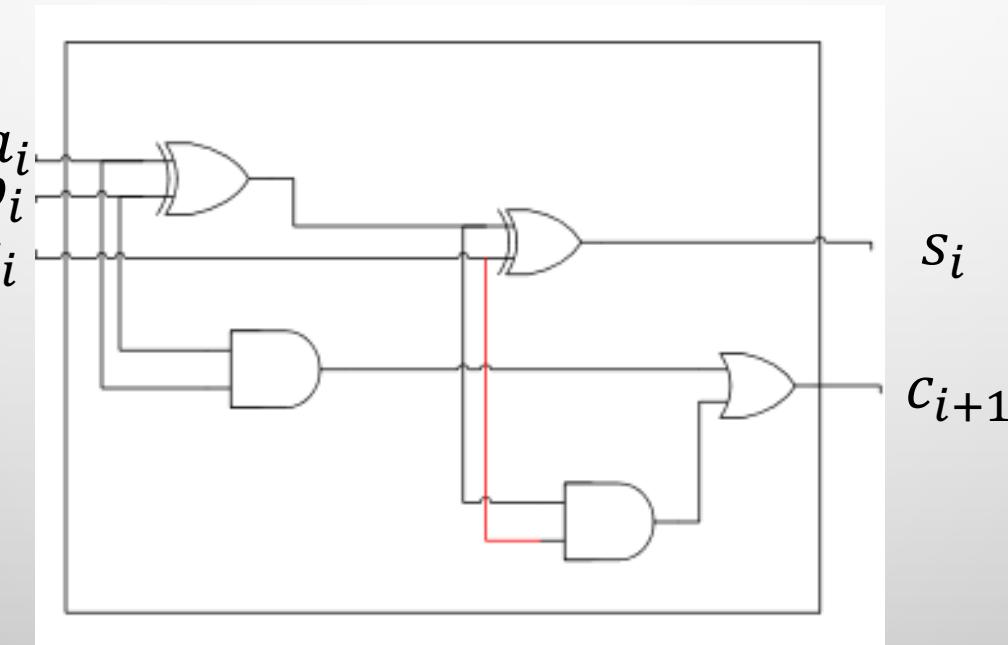
$a_i$	$b_i$	$c_i$	$s_i$	$c_{i+1}$
0	0	0	0	0
0	1	0	1	0
1	1	0	0	1
1	0	0	1	0
			<hr/>	
0	0	1	1	0
0	1	1	0	1
1	1	1	1	1
1	0	1	0	1

# Fonctions arithmétiques :

## Additionneur

$$s_i = a_i \oplus b_i \oplus c_i$$

$$\begin{aligned}c_{i+1} &= a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i \\&= (a_i \oplus b_i) \cdot c_i + a_i \cdot b_i\end{aligned}$$

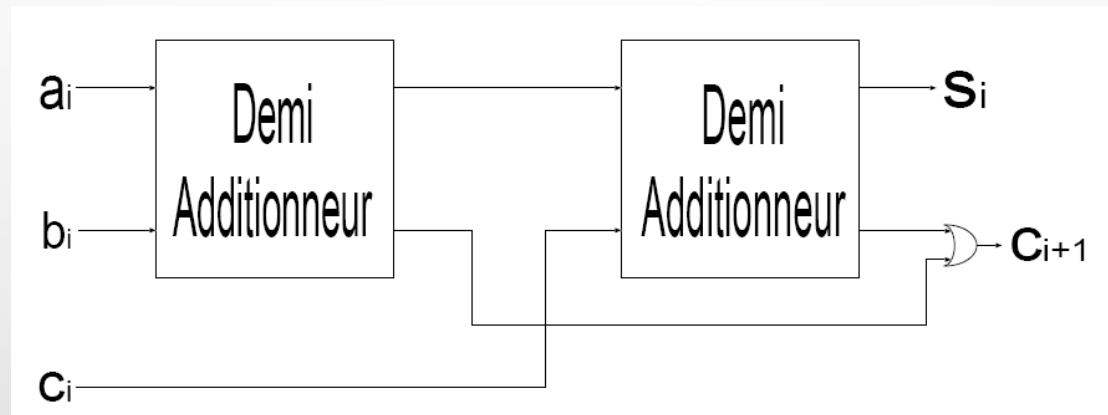


# Fonctions arithmétiques :

## Additionneur

Réalisation d'un additionneur 1 bit à l'aide de deux demi-additionneurs

- En schématique



$$s_{int} = a_i \oplus b_i$$

$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$r_{int} = a_i \cdot b_i$$

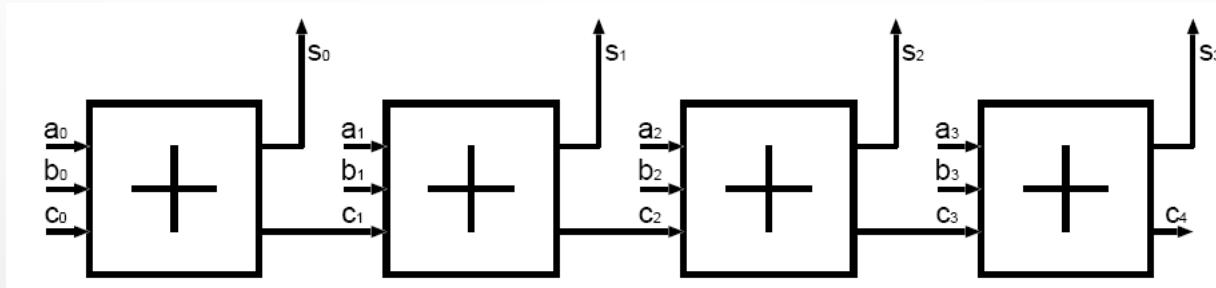
$$r_i = (a_i \oplus b_i) \cdot c_i$$

$$c_{i+1} = (a_i \oplus b_i) \cdot c_i + a_i \cdot b_i$$

# Fonctions arithmétiques :

## Additionneur

Additionneur 4 bits à retenue propagée ou série



- Solution intéressante car répétitive.
- le résultat de l'opération est disponible lorsque toutes les retenues, du poids le plus faible au poids le plus fort, ont été calculées
- d'où le nom d'additionneur à **retenue propagée**.
- Le temps de calcul est proportionnel à la taille des nombres manipulés donc au nombre d'étages du dispositif, soit  $n$ .

# Fonctions arithmétiques : Additionneur

## Temps de réponse d'un additionneur

- Les additionneurs sont des circuits physiques
- Chaque état correspond à un état énergétique
- Passage d'un état à un autre nécessite un apport d'énergie
  
- Temps de réponse prohibitif si nombreux étages
- Optimisation temporelle → solutions
  - ◆ Niveau logique: conception, équations....
  - ◆ Niveau électronique
  - ◆ Niveau géométrique (« layout »)
- Recherches toujours en cours : l'additionneur reste le circuit clé des circuits numériques

# Additionneur : Décimal Codé Binaire (BCD)

- Le code BCD (Binary Coded Decimal) a surtout été utilisé au début des machines à calculer.
- Il est parfois encore utilisé dans le système qui doivent afficher de l'information numérique à l'usager ou pour des systèmes qui manipulent de l'argent (Banque).
- Le code BCD encode le nombre à représenter de façon très directe. Chaque chiffre est encodé sur 4 bits. Les possibilités binaires de 10 à 15 ne sont pas utilisées.

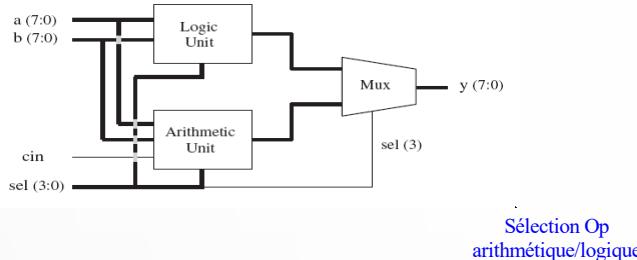
$$\begin{array}{r} 9 \quad 6 \quad 5 \\ \hline 1001 \quad 0110 \quad 0101 \end{array}$$

# Additionneur : Décimal Codé Binaire (BCD)

A	B	A plus B binaire	A plus B BCD
0000	0000	0000	0000
0001	-----	-----	1001
		1001	1 0000
		1010	1 0001
		1011	1 0010
		1100	1 0011
		1101	1 0100
		1110	1 0101
		1111	1 0110
		1 0000	1 0111
		1 0001	1 1000
1001	1001	1 0010	

Plus 0110

# Arithmetic Logic Unit (ALU)



sel	Operation	Function	Unit
0000	$y \leq a$	Transfer a	
0001	$y \leq a+1$	Increment a	
0010	$y \leq a-1$	Decrement a	
0011	$y \leq b$	Transfer b	
0100	$y \leq b+1$	Increment b	
0101	$y \leq b-1$	Decrement b	
0110	$y \leq a+b$	Add a and b	Arithmetic
0111	$y \leq a+b+cin$	Add a and b with carry	
1000	$y \leq \text{NOT } a$	Complement a	
1001	$y \leq \text{NOT } b$	Complement b	
1010	$y \leq a \text{ AND } b$	AND	
1011	$y \leq a \text{ OR } b$	OR	
1100	$y \leq a \text{ NAND } b$	NAND	
1101	$y \leq a \text{ NOR } b$	NOR	
1110	$y \leq a \text{ XOR } b$	XOR	
1111	$y \leq a \text{ XNOR } b$	XNOR	

Code Opératoire (mot de commande sur 3 bits)

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 -----
6 ENTITY ALU IS
7 PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8 sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9 cin: IN STD_LOGIC;
10 y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 -----
13 ARCHITECTURE dataflow OF ALU IS
14 SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16 ----- Arithmetic unit: -----
17 WITH sel(2 DOWNTO 0) SELECT
18 arith <=  a WHEN "000",
19          a+1 WHEN "001",
20          a-1 WHEN "010",
21          b WHEN "011",
22          b+1 WHEN "100",
23          b-1 WHEN "101",
24          a+b WHEN "110",
25          a+b+cin WHEN OTHERS;
```

26 ----- Logic unit: -----  
27 WITH sel(2 DOWNTO 0) SELECT  
28 logic <= NOT a WHEN "000",  
29 NOT b WHEN "001",  
30 a AND b WHEN "010",  
31 a OR b WHEN "011",  
32 a NAND b WHEN "100",  
33 a NOR b WHEN "101",  
34 a XOR b WHEN "110",  
35 NOT (a XOR b) WHEN OTHERS;  
36 ----- Mux: -----  
37 WITH sel(3) SELECT  
38 y <= arith WHEN '0',  
39 logic WHEN OTHERS;  
40 END dataflow;  
41 -----

# Multiplieur Combinatoire

- Premier multiplicateur
- Principe: suite d 'additions avec un des opérateurs successivement décalés

$$\begin{array}{r} A & 0 \ 1 \ 0 \ 1 \ + 5 \\ B & 1 \ 1 \ 1 \ 0 \ 1 \ + 13 \\ \hline & 0 \ 1 \ 0 \ 1 \\ \text{Plus} & 0 \ 0 \ 0 \ 0 \\ \hline & 0 \ 0 \ 1 \ 0 \ 1 \\ \text{Plus} & 0 \ 1 \ 0 \ 1 \\ \hline & 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \text{Plus} & 0 \ 1 \ 0 \ 1 \\ \hline & 1 \ 0 \ 0 \ 0 \ 0 \ 1 \quad +65 \end{array}$$

$S_0 = A \cdot B_0$

$S_1 = A \cdot B_1 \rightarrow$  décalage de 1

$S_2 = A \cdot B_2 \rightarrow$  décalage de 2

$S_3 = A \cdot B_3 \rightarrow$  décalage de 3

# Multiplieur Combinatoire

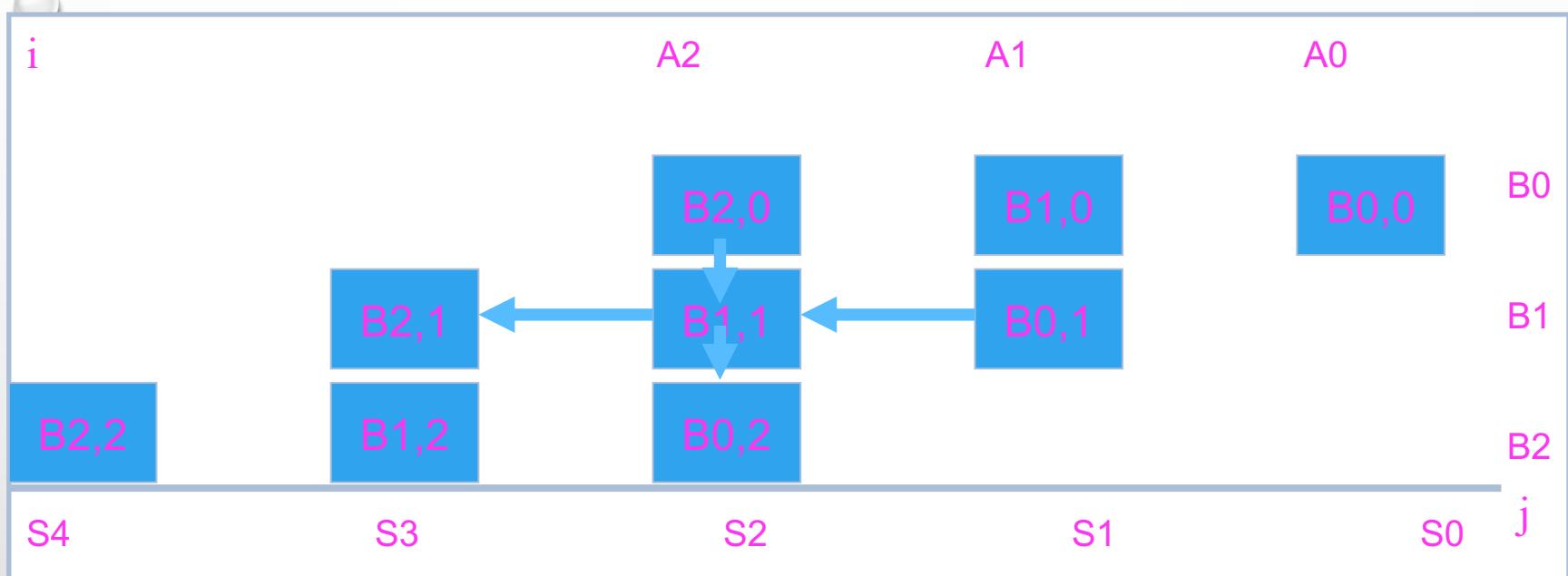


Table de multiplication:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

=> fonction ET

# Conception des systèmes numériques

## 4. Logique séquentielle Horloge, compteur

# Logique séquentielle

## Définition

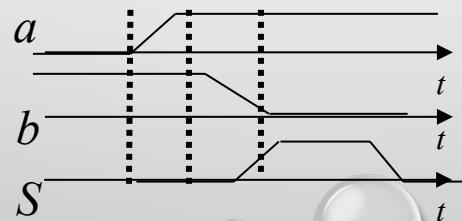
- On appelle circuit séquentiel un circuit pour lequel l'état des sorties à un instant donné  $t$  dépend à la fois de l'état des entrées et des états qu'avaient les sorties à l'instant  $t-1$ .
- On distingue:
  - ✓ les circuits séquentiels asynchrones pour lesquels il n'existe pas de référence de temps, c'est à dire que l'action des entrées est prise en compte dès leur changement d'état.
  - ✓ les circuits synchrones pour lesquels les entrées sont insensibles aux signaux qui leur sont appliqués sauf pendant un court intervalle de temps déterminé par un signal que l'on appelle une horloge.

Systèmes à logique séquentielle = Systèmes à événements discrets

# Logique séquentielle

- Dans les expressions booléennes, on suppose que la valeur de l'expression change instantanément à chaque changement des variables.
- Porte logique réelle : la sortie évolue continûment entre les niveaux H et L. Le changement de niveau prend un certain temps
- Le fonctionnement d'une porte réelle est continu. La considérer comme booléen n'est qu'une approximation utile
- En toute rigueur, les systèmes combinatoires n'existent pas!

Exemple:



$$S = ab\bar{}$$

# Logique séquentielle

## Hypothèses

- Le temps nécessaire à l'établissement de la valeur de sortie d'une porte logique est vu comme un simple retard → Principe de causalité
- La connaissance de l'évolution fine du niveau de signal logique n'est pas nécessaire
- Dans ces conditions, les variables d'une expression booléenne peuvent être remplacées par des fonctions du temps.

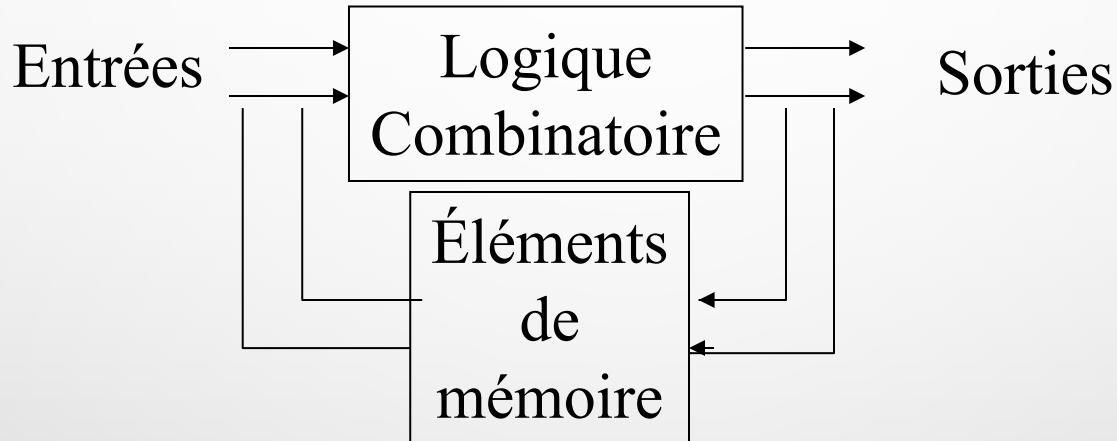
# Logique séquentielle : Exemple

Exemple : « paralléliser » les calculs en logique séquentielle

1. Si les débits de calculs sont 2 fois trop faibles en logique combinatoire, une solution consiste à mettre en parallèle 2 opérateurs et présenter alternativement les données impaires sur le premier et paires sur l'autre.
2. La logique séquentielle permet d'orienter correctement les données et de concaténer les résultats.
3. L'ordonnancement temporel et conditionnel des tâches permet de concevoir des algorithmes de machines à calculs génériques comme les automates et les processeurs.
4. Le séquencement nécessite une fonction propre à la logique séquentielle : la mémorisation. Celle-ci permet de fixer les données et les commandes de façon à les réutiliser dans<sup>54</sup> un ordre défini.

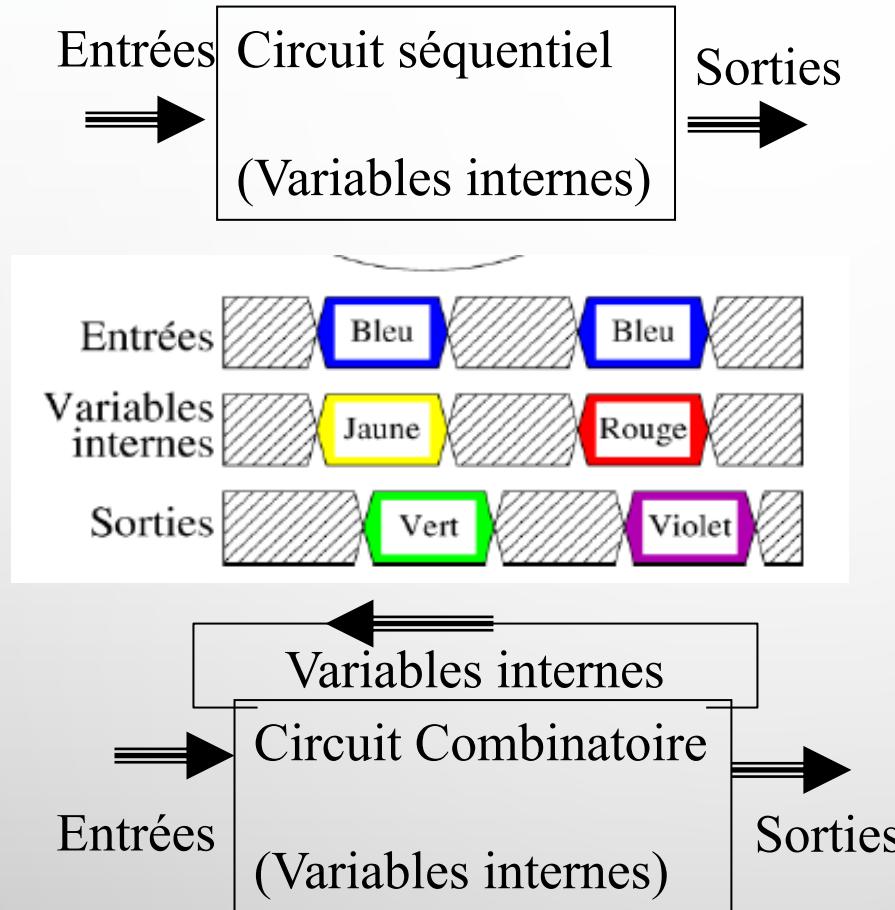
# Logique séquentielle

- Les systèmes séquentiels peuvent être vus comme des blocs de différentes formes qui sont des multivibrateurs



Généralement les sorties peuvent prendre seulement deux états stables  $Q$  et  $\bar{Q}$

# Logique séquentielle : variables d'états



Les valeurs des variables internes reflètent « l'état du système »

C'est le rebouclage qui réalise la fonction de mémorisation propre à la logique séquentielle

# Logique séquentielle : temps et synchronisation

- La structure du circuit de logique séquentielle possède donc un circuit combinatoire calculant les variables internes et recevant les entrées:
  - les temps de propagation dans un circuit combinatoire sont extrêmement variables et dispersifs
  - Dans ces conditions, il est difficile de faire fonctionner le dispositif précédent d'une façon fiable car **chaque variable interne et chaque sortie ont leurs propres temps de propagation**
  - Une méthode largement répandue pour l'évolution des calculs consiste à **synchroniser les calculs** ainsi les variables internes et les sorties sont gelées dans une **mémoire**, généralement une **bascule**.

# Logique séquentielle : temps et synchronisation

- Les sorties sont mémorisées car elles sont potentiellement utilisées comme entrées d'autres circuits séquentiels
- La mise à jour des mémoires, ou échantillonnage des résultats, peut se faire d'une façon synchrone au rythme d'un signal de commande périodique : l'horloge.
- Dans ce cas, l'instant d'échantillonnage correspond à une transition montante ou descendante du signal d'horloge de rapport cyclique  $\alpha$

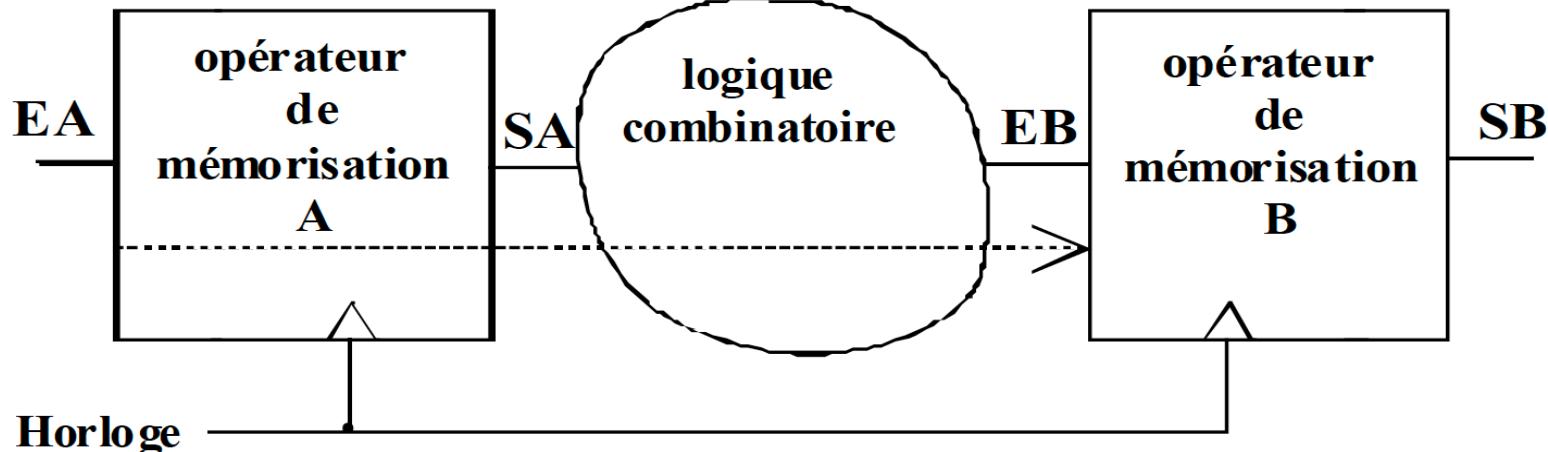
# Logique séquentielle : temps et synchronisation

Le chemin le plus lent d'un circuit combinatoire s'appelle « chemin critique ». Si  $T_h$  est la période d'horloge et  $T_{crit}$  est le temps de propagation du chemin critique, alors il suffit de respecter

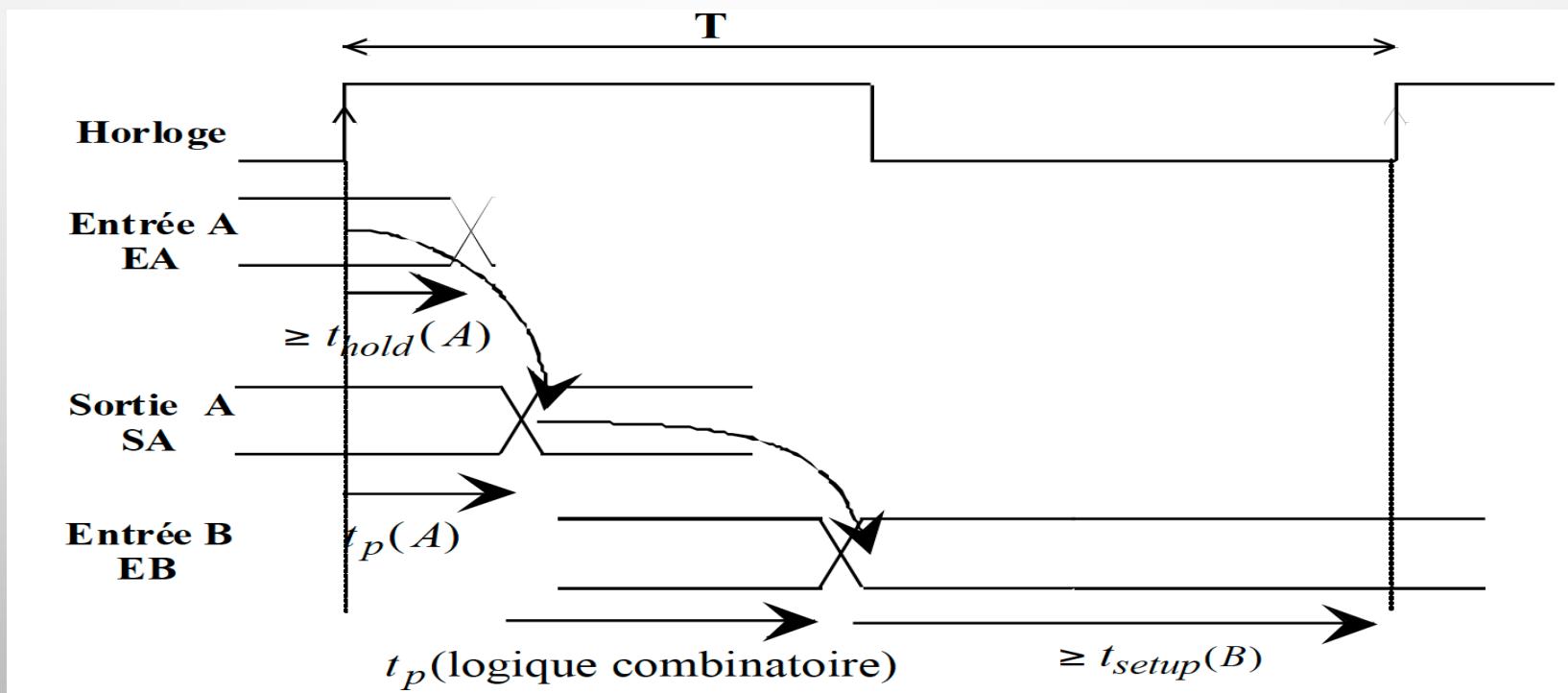
$$T_h > T_{crit}$$

Le calcul du chemin critique se fait dans les conditions d'utilisation les plus dégradées

→ c'est-à-dire un procédé technologique sous-optimal : comme par exemple une tension d'alimentation  $V_{dd}$  haute et une température de jonction élevée



Les 2 fonctions de mémorisation ne sont pas, en pratique, nécessairement distinctes.  
Il peut s'agir de la même bascule (cas du rebouclage d'une sortie sur l'entrée).



# Logique séquentielle : temps et synchronisation

Pendant une période T de l'horloge, il faut prendre en compte :

- Les temps de propagation de l'élément de mémorisation A (bascule) et de la logique combinatoire,
- Le temps de maintien de l'entrée de A,  $EA$ , après le premier front actif de l'horloge,
- Le temps de prépositionnement de l'entrée  $EB$  du second élément de mémorisation B, avant le front actif suivant de l'horloge.

En pratique, les différents temps de propagation d'une bascule sont toujours supérieurs au temps de maintien, quelle que soit la charge de la bascule. La période T du signal d'horloge doit donc vérifier pour tous les chemins de propagation la relation suivante :

$$T_h > t_{p \max}(A) + t_{p \max}(\text{logique combinatoire}) + t_{\text{setup}}(B)$$

Le chemin critique est le chemin de propagation «le plus long » du circuit, c'est-à-dire celui qui impose la contrainte la plus forte sur T.

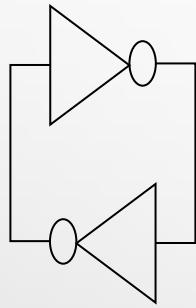
# Élément de base : mémorisation

Le système de base spécifique à la logique séquentielle est le point mémoire. Il existe différentes technologies pour créer le point mémoire.

1. La bascule RS est la brique de base des systèmes séquentiels
2. La « bascule D » est un composant de mémorisation pour un seul point mémoire.
3. La mémoire RAM Random Access Memory est un ensemble de points mémoires regroupés dans une matrice.
4. L'accès à la RAM ne permet pas d'accéder à tous les points mémoires en même temps, mais à un seul. Un mécanisme d'adressage est donc nécessaire pour sélectionner un point mémoire qui dispose ainsi de sa propre « adresse »

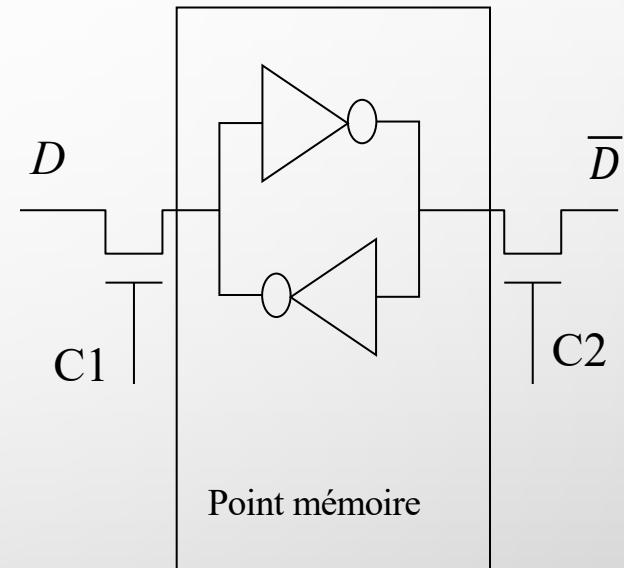
# Élément de base : mémorisation

Exemple simple de structure bistable : 2 inverseurs tête bêche



Point mémoire bistable

Ce système est extrêmement stable  
→ Toutefois il faut l'initialiser ou le commander

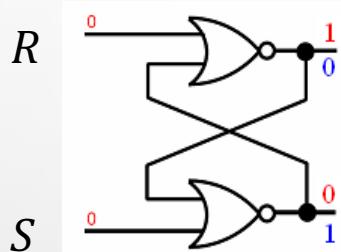


Commande écriture ou lecture C1/C2

# Logique séquentielle : la bascule RS

- La bascule RS (RS Latch)

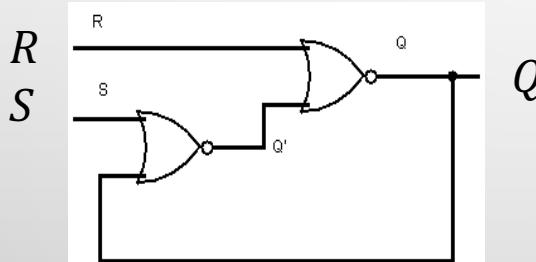
2 portes NOR  $(\overline{a + b})$



$Q$

$\bar{Q} = Q'$

$S$	$R$	$Q$	Fonction	Complémentarité
0	0	$Q_I$	Mémorisation	$Q' = \bar{Q}$
0	1	0	Reset (mise à 0)	$Q' = \bar{Q}$
1	0	1	Set (mise à 1)	$Q' = \bar{Q}$
1	1	Combinaison ambiguë		$Q' \neq \bar{Q}$



$Q$

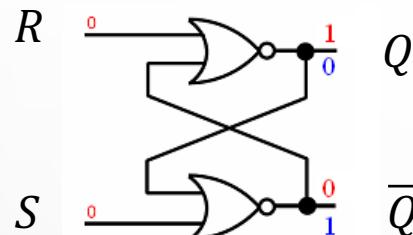
Cet état force à la fois à 1 et 0 la sortie  $Q$

$Q'$  est une variable d'état

# Logique séquentielle : les bistables

- La bascule RS      2 portes NOR       $(\overline{a + b})$

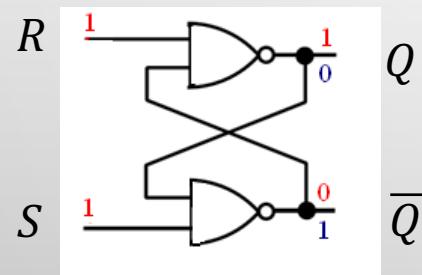
La bascule RS peut être construite à partir de portes NAND



R	S		Q	$\bar{Q}$
0	0		Q	$\bar{Q}$
0	1		1	0
1	0		0	1
1	1			

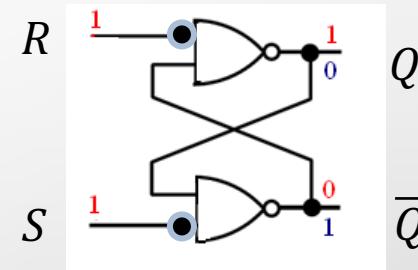
Mémoire  
Set  
Reset  
état interdit

- 2 portes NAND       $(\overline{a \cdot b})$



R	S		Q	$\bar{Q}$
1	1		Q	$\bar{Q}$
1	0		1	0
0	1		0	1
0	0			

état interdit



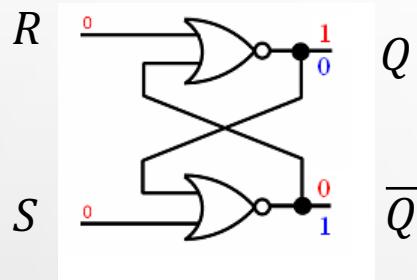
R	S		Q	$\bar{Q}$
0	0		Q	$\bar{Q}$
0	1		1	0
1	0		0	1
1	1			

état interdit

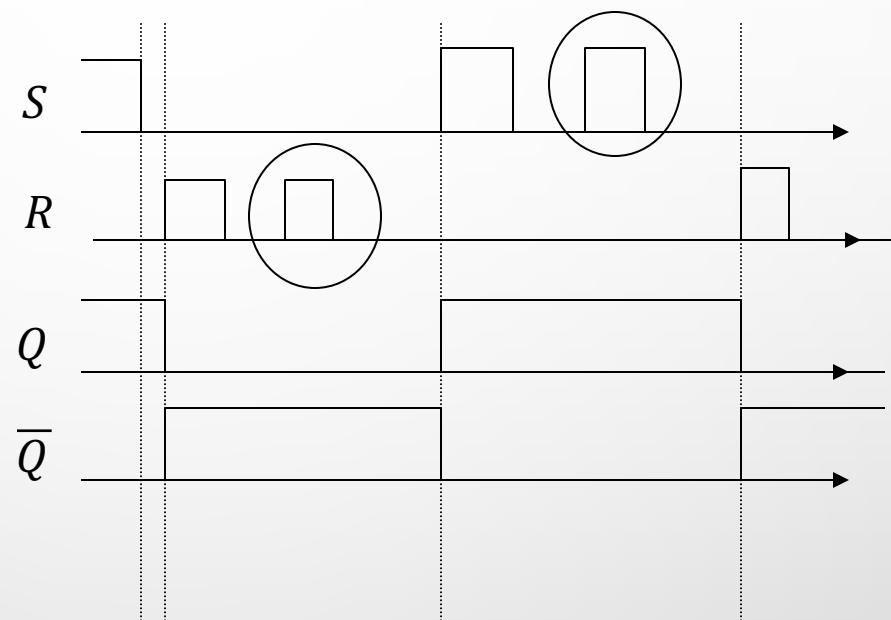
# Logique séquentielle

- La bascule RS = bascule de base asynchrone

2 portes NOR  $(\overline{a + b})$



R	S		Q	$\bar{Q}$	
0	0		Q	$\bar{Q}$	Mémoire
0	1		1	0	Set
1	0		0	1	Reset
1	1				état interdit

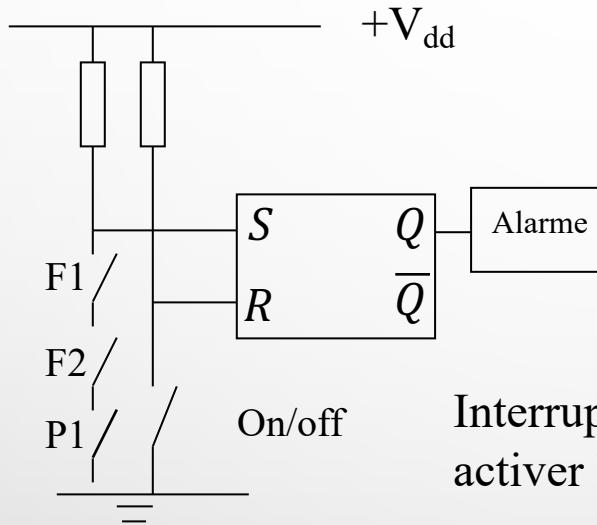


# Logique séquentielle

## Les éléments séquentiels de base

- La bascule RS – exemple d'utilisation : alarme de maison

Interrupteurs  
de portes ou  
de fenêtres



Interrupteur fermé pour activer l'alarme

$R$	$S$	$Q$	$\bar{Q}$	
0	0	$Q$	$\bar{Q}$	Mémoire
0	1	1	0	Set
1	0	0	1	Reset
1	1			état interdit

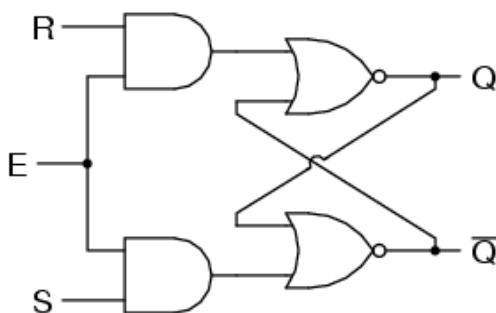
Les interrupteurs sont fermés quand les fenêtres ou les portes sont fermées  $\rightarrow S=0$   
L'ouverture d'un interrupteur  $\rightarrow S \rightarrow 1$

# Bascule RS en VHDL

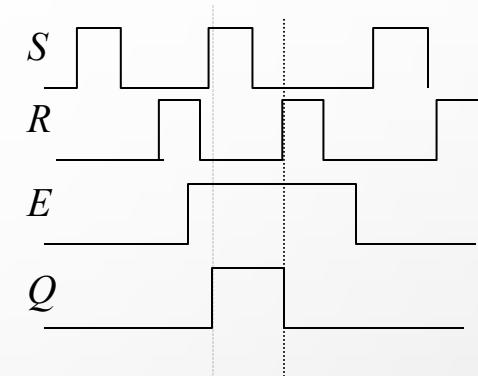
```
Entity RS is port (
    R,S : in bit;
    Q : out bit);
end RS;
architecture df of RS is
signal etat : bit;
begin
    q <= etat;
    with R&S select -- l'opérateur & concatène les signaux R et S
        etat <=  '1' when "01",
                  '0' when "10",
                  etat when "00", -- mode mémoire explicite
                  'x' when "11";
end df;
```

# Logique séquentielle: RS-latch

- La bascule RS-latch (verrouillée)



E	S	R	Q	$\bar{Q}$
0	0	0	latch	latch
0	0	1	latch	latch
0	1	0	latch	latch
0	1	1	latch	latch
1	0	0	latch	latch
1	0	1	0	1
1	1	0	1	0
1	1	1		

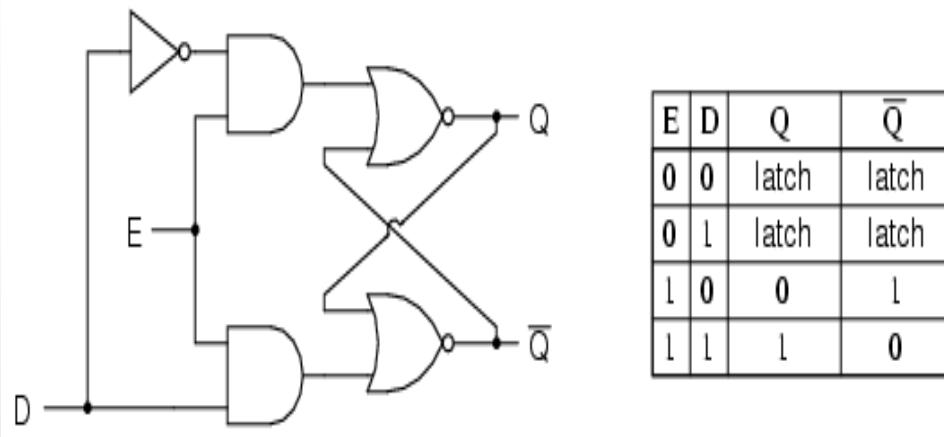


On peut contrôler les opérations d'entrées grâce à une entrée de type Enable  $E$ .

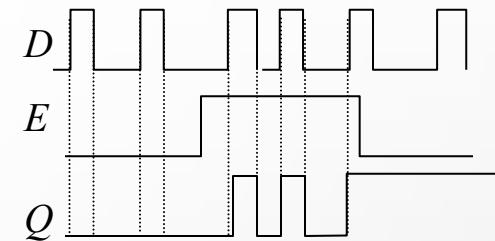
→ Les 2 portes NAND d'entrée permettent d'autoriser ou d'inhiber les actions des entrées  $R$  et  $S$

# Logique séquentielle : D latch

- La bascule D latch (verrouillée)



E	D	Q	$\bar{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0



Latch = mémorisation

La bascule D latch n'a plus qu'une entrée D et un verrouillage.

Il n'y a plus l'effet indéterminé ( $R=S=1$ )

La mémorisation (l'état ( $R=S=0$ ) de la bascule RS) est possible uniquement par l'entrée  $E=0$

Quand  $E=1$  la sortie  $Q$  suit l'entrée  $D$

→ Application

Les bascules D Latch sont souvent utilisées pour lire et regrouper des mots<sub>71</sub> de 4, 8 ou 16... bits quand  $E=1$  → bus de données

# Logique séquentielle : D latch

- La bascule D latch (verrouillée)

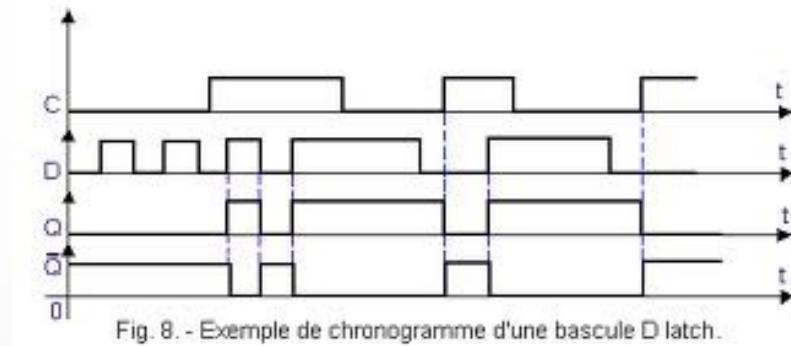
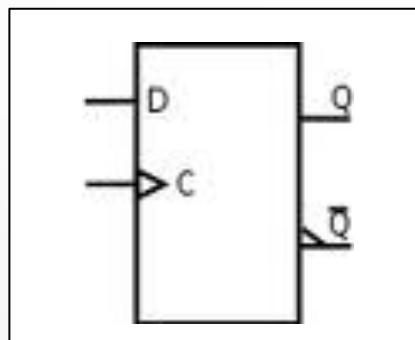
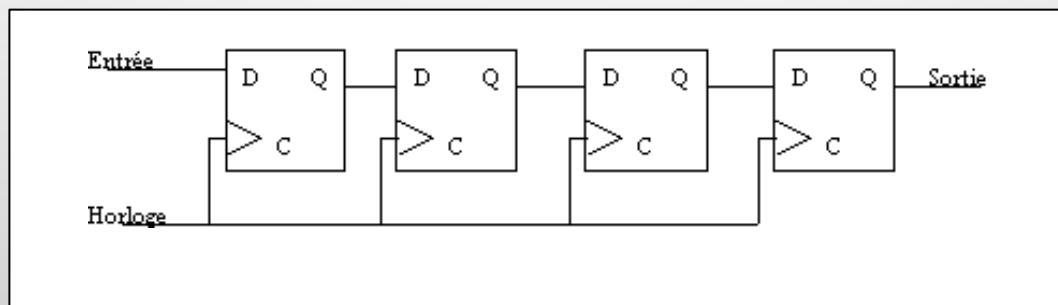


Fig. 8. - Exemple de chronogramme d'une bascule D latch.

$$\text{Equation de la Bascule D latch : } Q_{n+1} = D$$

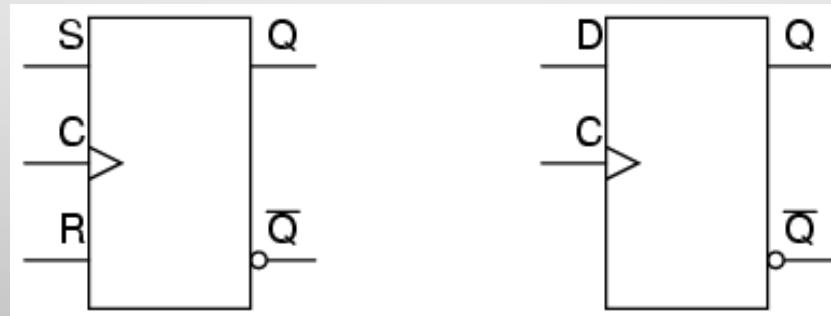
Exemple d'application



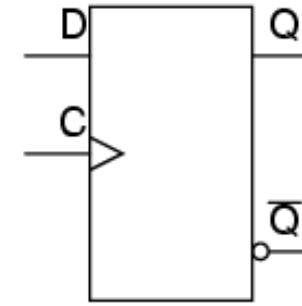
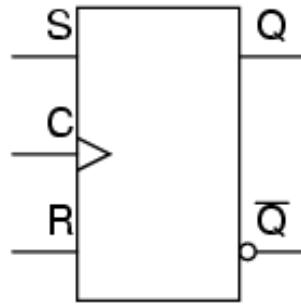
En 4 cycles d'horloge, un mot de 4 bits est mémorisé.

# Logique séquentielle : horloge

- Dans de nombreuses situations, il est nécessaire de synchroniser des opérations de différents circuits
  - Contrôler précisément les changements d'états
  - De nombreux multivibrateurs bistables sont construits pour changer d'états lors de l'application d'un signal horloge ('trigger')
  - Ils sont appelés bistables flip-flops
  - Le changement d'états s'effectue soit sur front montant soit sur front descendant du signal l'horloge
  - Systèmes Flip-flops = bascules à verrouillage avec Enable=Horloge(Clock)



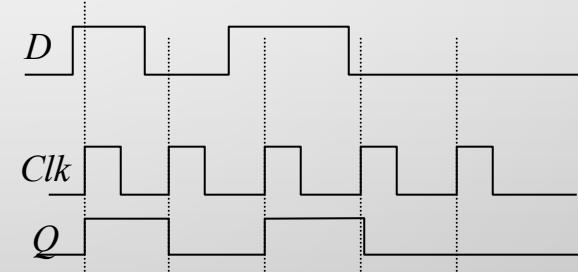
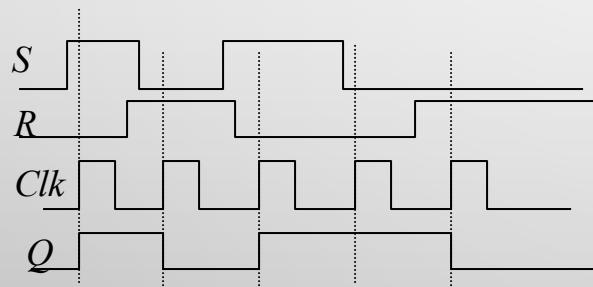
# Logique séquentielle



$S$	$R$	$Clk$	$Q_n$	$\overline{Q_n}$	
0	0	$\uparrow$	$Q_{n-1}$	$\overline{Q_{n-1}}$	Mémoire
0	1	$\uparrow$	0	1	Reset
1	0	$\uparrow$	1	0	Set
1	1	$\uparrow$	0	0	Ambiguité

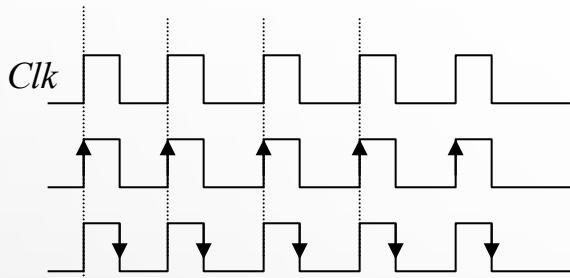
Enable devient une horloge

$D_{n-1}$	$Clk$	$Q_n$	$\overline{Q_{n+1}}$	
0	$\uparrow$	0	1	Reset
1	$\uparrow$	1	0	Set



# Horloge ou fronts d'horloge

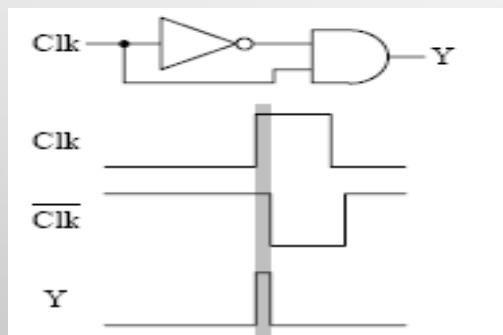
- Horloge : signification



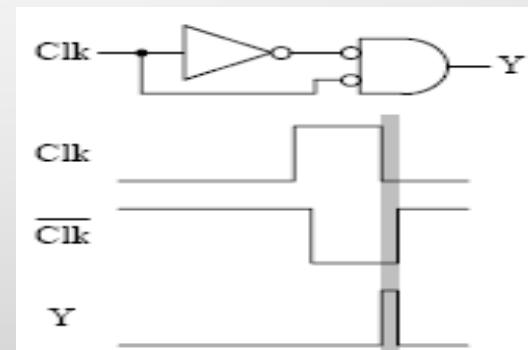
Front d'horloge: en pratique cela signifie que la commande *Clk* ou E est activée pendant un temps réduit par rapport à la période de l'horloge

- Horloge : technologie

Détecteur de front montant



Détecteur de front descendant



# Horloge ou fronts d'horloge

- Horloge: signification



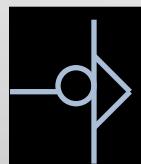
Bascule sensible à un état (level sensitive)  
Ici état 1



Bascule sensible à un état (level sensitive)  
Ici état 0



Bascule sensible à un front (edge triggered)  
Ici front montant



Bascule sensible à un front (edge triggered)  
Ici front descendant

# Horloge et VHDL

VHDL ne contient pas le concept de signal d'horloge.

Solution introduire une instruction "WAIT" dans un processus

Exemple :

```
entity basc_synchronise is port (
    clock : in bit;
    commande : in bit_vector( ... );
    q: out bit);
end basc_synchronise;
architecture fsm of basc_synchronise is
signal etat : bit;
begin
    q <= etat;
    process
        begin
            wait until (clock = '1')
            case etat is
                when '0' =>
                    -- conditions de la transition '0'->'1'
                when '1' =>
                    -- conditions de la transition '1'->'0'
            end case;
        end process;
    end fsm
```

# Horloge et VHDL

- Une variante qui remplace l'instruction « WAIT » par une liste de sensibilité du processus et un test sur l'existence d'une transition du signal d'horloge et le niveau qui suit cette transition

Exemple :

```
architecture d_primitive1 of d_edge is
begin
    process(hor) -- Le process ne « réagit » qu'au signal hor.
    begin
        If (hor'event and hor = '1') then --- deux conditions
        s <= d ;
        end if;
    end process;
end d_primitive1
```

# La bascule JK

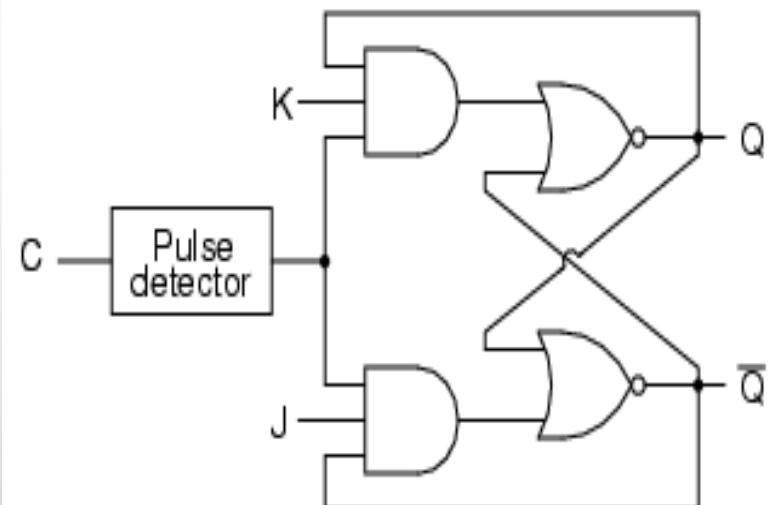
## ■ La bascule JK

Les bascules RS souffrent de leur état d'entrée ambigu ( $R=1$  et  $S=1$  pour une bascule RS NOR ou  $R=0$  et  $S=0$  pour une bascule RS NAND)

Les bascules D lèvent l'ambiguïté mais elles ne possèdent plus qu'une entrée.

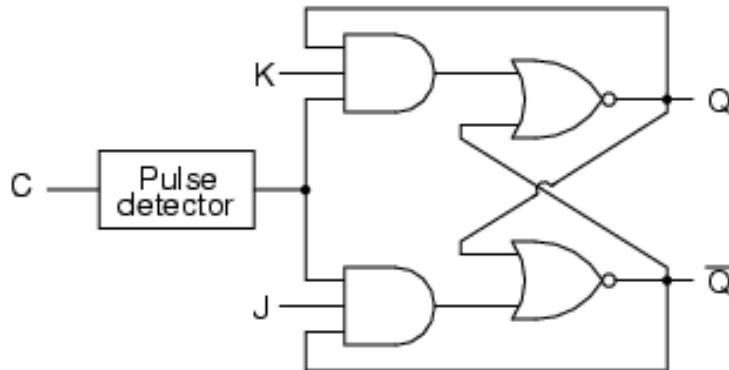
La structure de la bascule JK  
lève l'ambiguïté sur l'état  
incertain et conserve 2 entrées

$J \rightarrow$  correspond à l'entrée  $S$   
 $K \rightarrow$  correspond à l'entrée  $R$



# Logique séquentielle

## ■ La bascule JK



Etats	J	K	Clk	$Q_n$	$\overline{Q_n}$	
1	0	0	$\uparrow$	$Q_{n-1}$	$\overline{Q_{n-1}}$	Mémoire
2	0	1	$\uparrow$	0	1	Reset
3	1	0	$\uparrow$	1	0	Set
4	1	1	$\uparrow$	$\overline{Q_{n-1}}$	$Q_{n-1}$	Interrupteur à bascule (Toggle)

Les 2 entrées sont dites ‘interverrouillées’

→ Si la sortie est ‘set’ ( $Q=1$ ) l’entrée  $J$  est inhibée par  $\overline{Q_n} = 0$  à travers la porte ET →  $K=1$  impose le Reset  $Q=0$

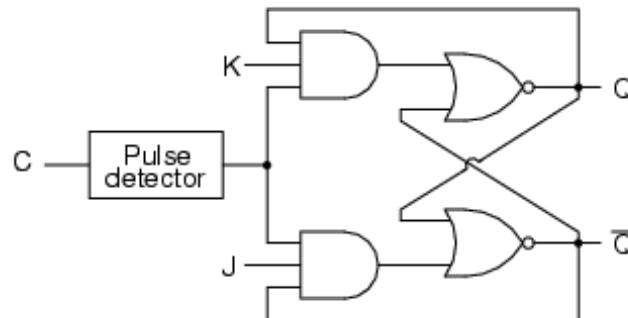
→ Si le circuit est ‘reset’, l’entrée  $K$  est inhibée par le 0 de la sortie  $Q$  à travers la porte ET →  $J=1$  impose le Set de la sortie  $Q=1$

L’entrée  $J$  intervient uniquement quand le circuit est ‘reset’

L’entrée  $K$  intervient uniquement quand le circuit est ‘set’.

# Logique séquentielle

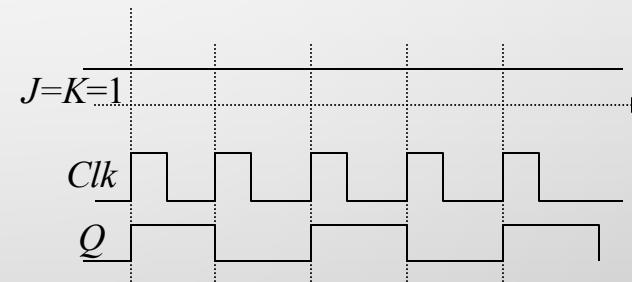
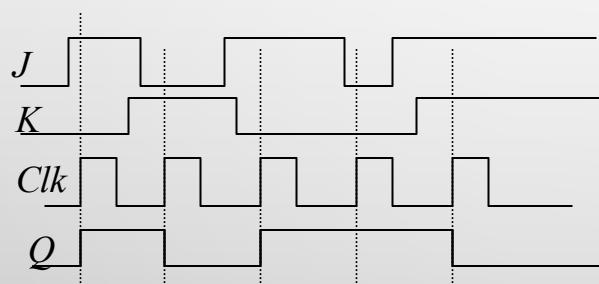
- La bascule JK



$J$	$K$	$Clk$	$Q_n$	$\bar{Q}_n$	
0	0	$\uparrow$	$Q_{n-1}$	$\bar{Q}_{n-1}$	Mémoire
0	1	$\uparrow$	0	1	Reset
1	0	$\uparrow$	1	0	Set
1	1	$\uparrow$	$\bar{Q}_{n-1}$	$Q_{n-1}$	Interrupteur à basculement (Toggle)

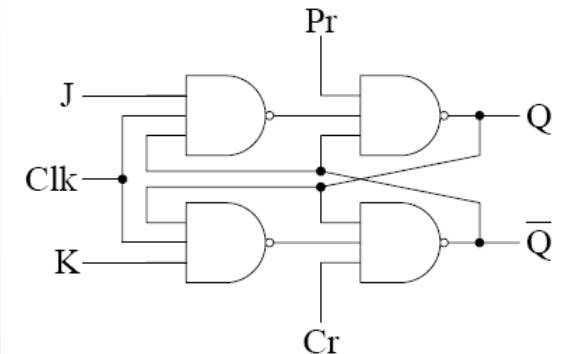
Quand les entrées deviennent telles que  $J=K=1$

→ Un front de l'horloge fait commuter les sorties telles qu'un état de sortie ( $Q=1$  et  $\text{not}(Q)=0$ ) commute à ( $Q=0$  et  $\text{not}(Q)=1$ ) et inversement.



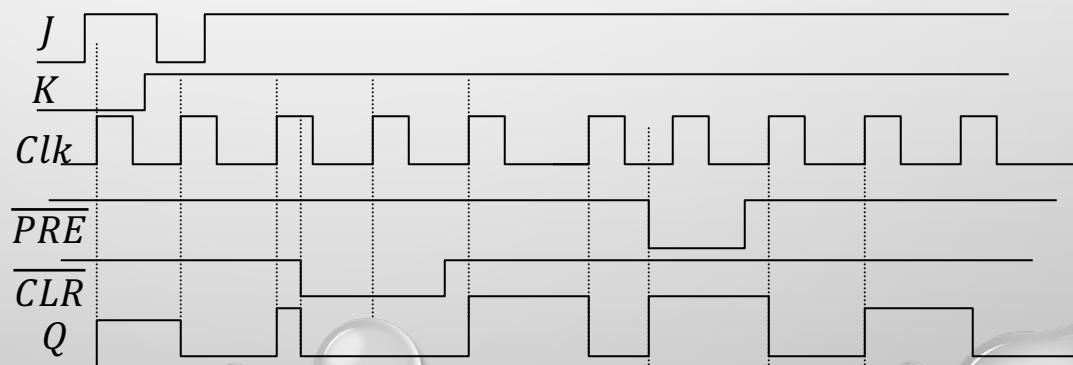
# Logique séquentielle

- La bascule JK + entrées asynchrones



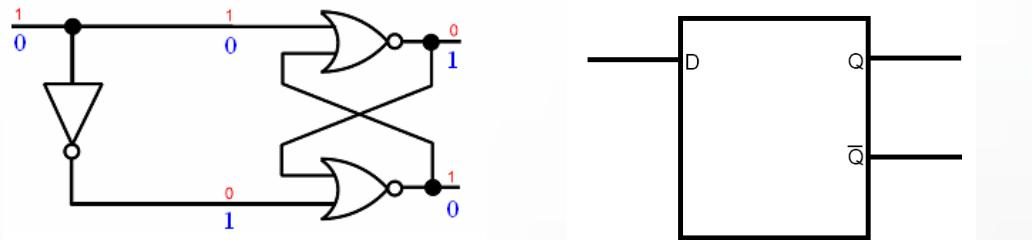
$Pr$	$Cr$	$Q$
1	1	$Q$
0	1	1
1	0	0

Les entrées asynchrones (prioritaires devant  $Clk$ )  $Pr$  (Preset) et  $Cr$  (Clear) permettent d'assigner l'état initial de la bascule, par exemple juste après la mise sous tension pour éviter tout aléa. En fonctionnement synchrone avec  $Clk$ , ces deux entrées doivent être maintenues à 1.

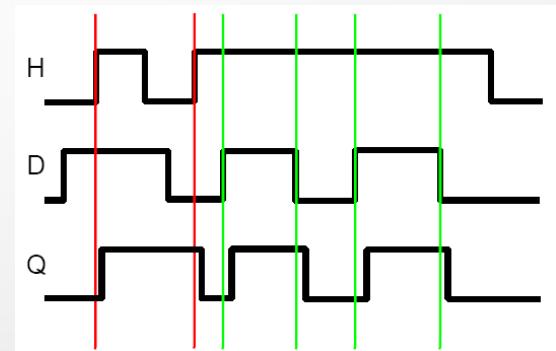
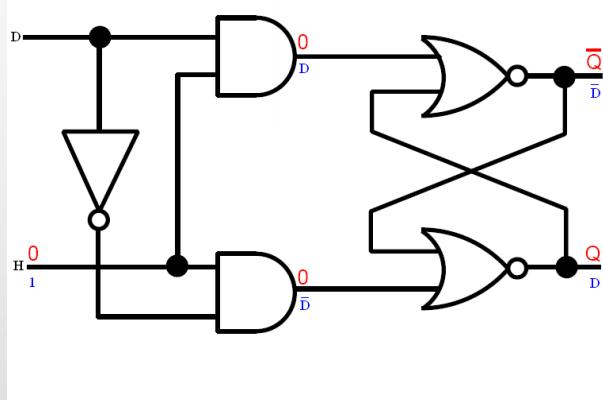


# Logique séquentielle

- La bascule D



- La bascule D active sur niveau (Dlatch)



Si  $H = 0 \rightarrow Q_n = Q_{n-1}$  Fonction mémoire

Si  $H = 1 \rightarrow Q_n = S + \bar{R}Q_{n-1} = D_{n-1}$  Fonction recopie de l'entrée

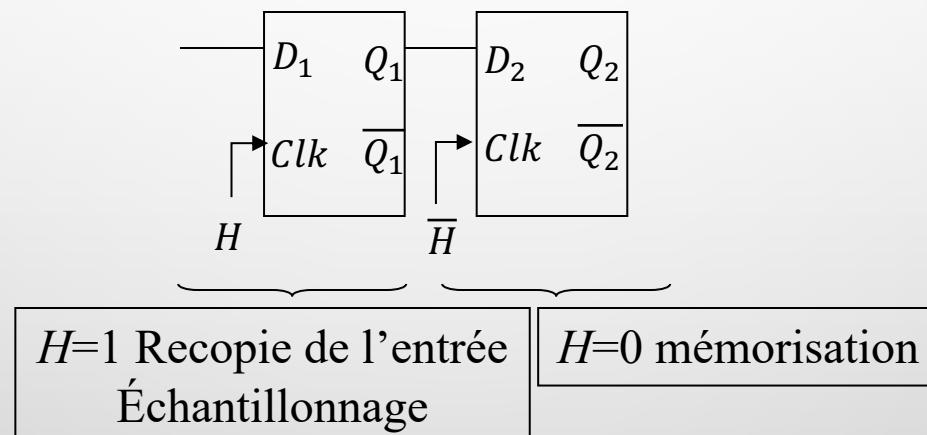
$D$	$H$	$Q_{n+1}$	$\bar{Q}_{n+1}$
X	0	$Q_n$	$\bar{Q}_n$
0	1	0	1
1	1	1	0

# Logique séquentielle

Pour des systèmes à plusieurs bascules, la synchronisation des variables internes et des sorties permet de fiabiliser les calculs.

La bascule doit pouvoir réaliser 2 fonctions :  
l'échantillonnage et la mémorisation

Ces 2 fonctions peuvent être obtenues avec 2 Dlatches en cascade

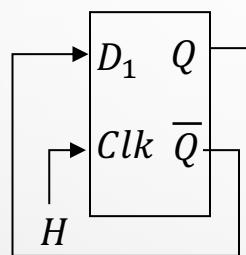


# Logique séquentielle

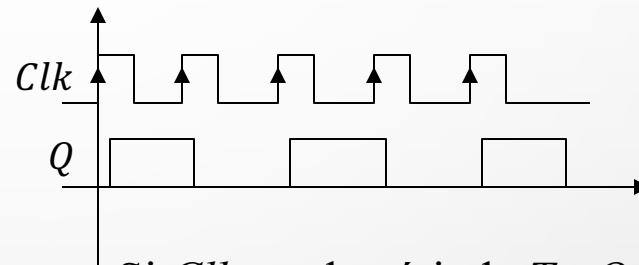
## Autre Exemple Conditions d'utilisation de la bascule D

Fonctionnement: soit une bascule à une entrée d'horloge et la sortie change d'état à chaque niveau haut de l'horloge.

$$Q_n = \overline{Q_{n-1}}$$



Si  $Clk = 1$  pendant  $T_h/2$  et que  $T_h$  est grand → il risque d'y avoir des instabilités via  $\overline{Q}$



Si  $Clk$  est de période  $T_h$ ,  $Q$  est de période  $2T_h$  → c'est un diviseur de fréquence.

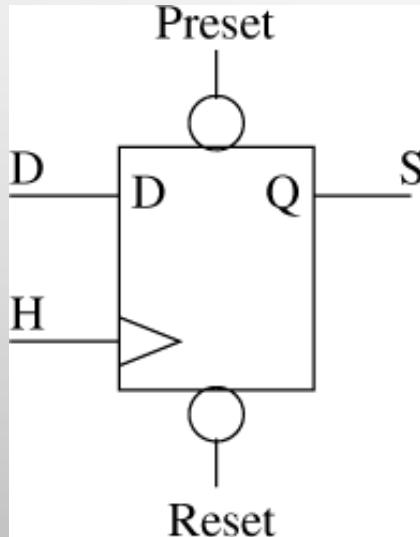


D'où l'intérêt de déclencher sur **fronts montants ou descendants** car la commande revient rapidement à 0

# Logique séquentielle

- La bascule D active sur front (D Flip Flop), description VHDL

- La bascule D active sur front (D Flip Flop) avec fonctions Preset et Reset

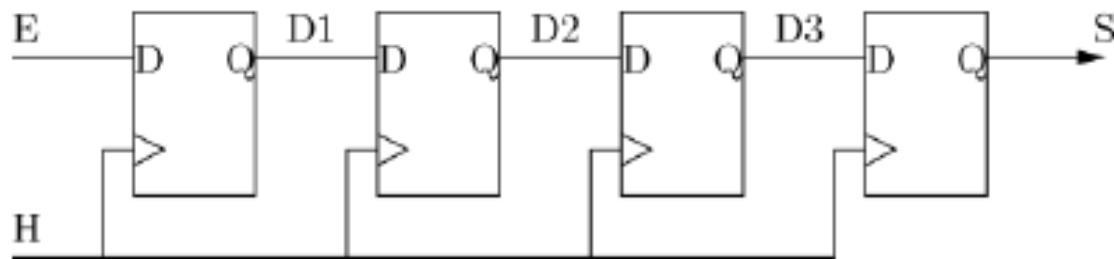


```
entity bascule is
port (      d, clk : in std_logic;
            q : out std_logic );
end entity bascule;
architecture comport of bascule is
begin
stockage : process(d,clk) is
begin
    if clk='1' and clk'event then
        q <= d;
    end if;
end process stockage;
end architecture comport;
```

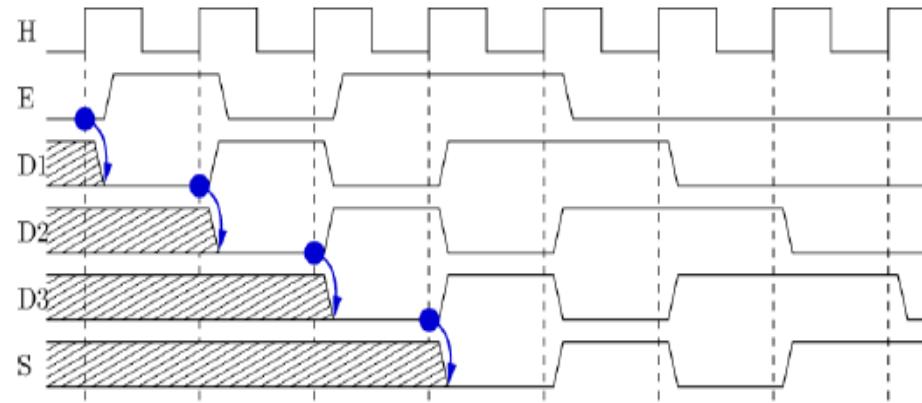
D	H	Preset	Reset	Q	Etat
0	↑	1	1	0	échantillonnage
1	↑	1	1	1	
X	0	1	1	Q	mémorisation
X	1	1	1	Q	
X	X	0	1	1	forçage à 1
X	X	1	0	0	forçage à 0

# Exemples fondamentaux – registre à décalage

A chaque front d'horloge, le contenu de chaque bascule amont est décalé dans la bascule aval.



Après  $n$  fronts montants d'horloge, la première valeur rentrée se retrouve en sortie des  $n$  bascules



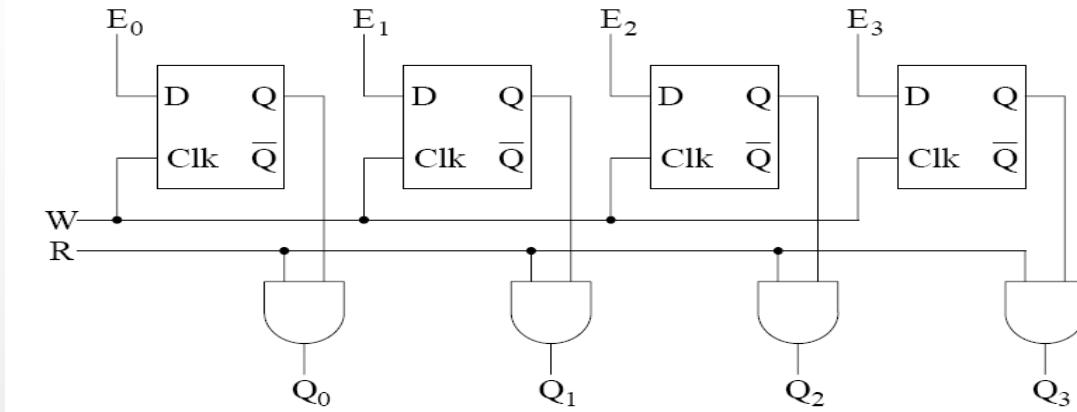
Le fonctionnement correct du registre impose d'avoir un temps de propagation  $T_{co}$  supérieur au temps de maintien  $T_h$ . Cette condition est toujours garantie par les constructeurs de bascules.

# Exemples fondamentaux – registre à décalage

Le registre à décalage est une structure importante de la logique séquentielle qui permet de réaliser beaucoup d'opérations élémentaires :

- Passage d'un format série à un format parallèle : les bits rentrent en série et les N bits du registre sont les sorties.
- Passage d'un format parallèle à un format série : Les bascules sont initialisées par un mot d'entrée et la sortie s'effectue sur la dernière bascule.
- Recherche d'une chaîne de bits particulière : les sorties des bascules sont comparées avec la chaîne de référence.
- Et encore : générateur de nombres pseudo aléatoires (LFSR Linear Feedback shift register), filtres numériques RIF, ...

# Exemples fondamentaux – Registre à décalage



Un registre permet la mémorisation de  $n$  bits soit  $n$  bascules, mémorisant chacune un bit. L'information est conservée et disponible en lecture.

En synchronisme avec le signal d'écriture W

Le registre mémorise les données présentes sur les entrées E<sub>0</sub>, E<sub>1</sub>, E<sub>2</sub> et E<sub>3</sub>.

Elles sont conservées jusqu'au prochain signal de commande W.

Dans cet exemple, les états mémorisés peuvent être lus sur les sorties Q<sub>0</sub>, Q<sub>1</sub>, Q<sub>2</sub> et Q<sub>3</sub> en coïncidence avec un signal de validation R.

Lorsque ces sorties sont connectées à un bus, les portes ET en coïncidence avec ce signal de lecture sont disponibles.

# Logique séquentielle

## Exemples fondamentaux de la logique séquentielle synchrone – description VHDL d'un registre de mémorisation

```
entity reg8generic is
generic (N : natural := 8);
port (      d :in std_logic_vector(N-1 downto 0);
            clk :in std_logic;
            q: out std_logic_vector(N-1 downto 0)
);
end entity reg8generic;

architecture comport of reg8generic is
begin
stockage : process(clk,d) is
begin
    if (clk='1' and clk'event) then
        q <= d;
    end if;
end process stockage;
end architecture comport;
```

## Exemples fondamentaux – compteurs

Un compteur est un ensemble de  $n$  bascules interconnectées par des portes logiques. Elles peuvent donc générer et mémoriser des mots de  $n$  bits.

- Au rythme d'une horloge, elles peuvent décrire une séquence déterminée, occuper une suite d'états binaires.
- Il ne peut y avoir au maximum que  $2^n$  combinaisons. Ces états restent stables et accessibles entre les impulsions d'horloge.
- Le nombre total  $N$  des combinaisons successives est appelé le modulo du compteur. On a  $N \leq 2^n$ . Si  $N < 2^n$  un certain nombre d'états ne sont jamais utilisés.
- La synthèse d'un compteur consiste à définir les commandes des bascules pour définir l'ordre du cycle prévu

# Exemples fondamentaux – compteurs

Les compteurs binaires peuvent être classés en deux catégories :

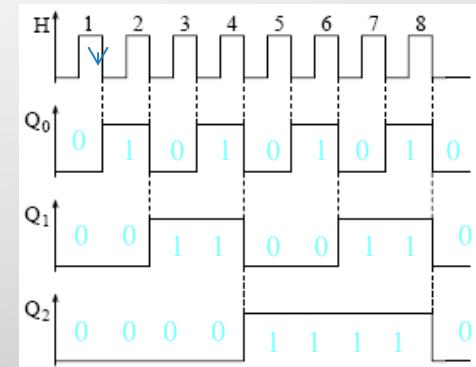
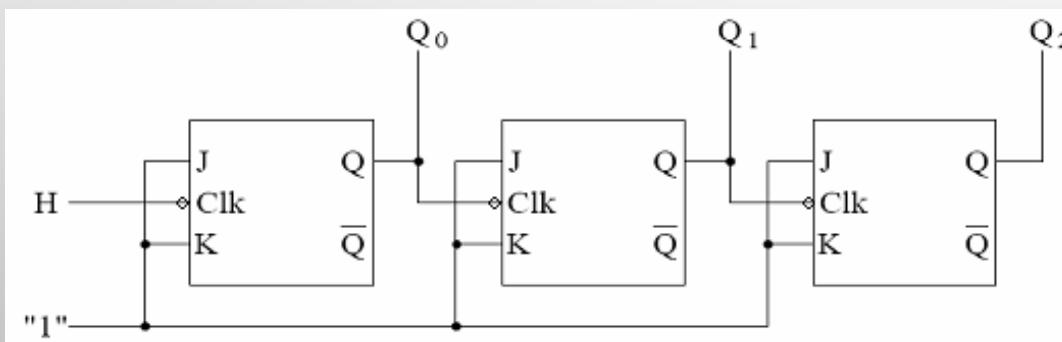
- Les compteurs asynchrones : l'ordre de changement d'états des bascules se réalise en cascade
- Les compteurs synchrones : le signal d'horloge synchronise toutes les bascules

De plus, on distingue les compteurs qui peuvent être réversibles également appelés compteurs-décompteurs.

# Logique séquentielle : compteur asynchrone

- Ils sont essentiellement utilisés dans la division de fréquence
- Un compteur asynchrone est constitué de  $n$  bascules J-K fonctionnant en mode Trigger ( $Q_n = \overline{Q_{n-1}}$  car  $J=K=1$ )
- Le signal d'horloge n'est reçu que par le premier étage (bascule LSB : Least Significant Bit). Pour chacune des autres bascules, le signal d'horloge est fourni par une sortie de la bascule de rang immédiatement inférieur.

Bascules JK sur front descendant



# Logique séquentielle

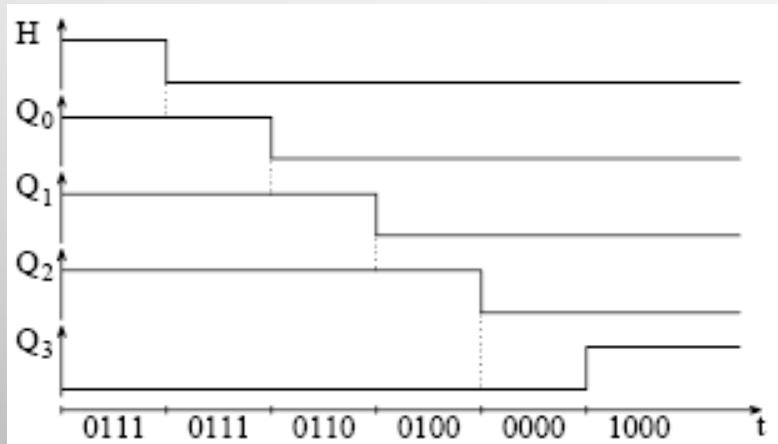
## Inconvénients des compteurs asynchrones

Comme chaque bascule a un temps de réponse

→ Compteur asynchrone = le signal d'horloge ne parvient pas simultanément sur toutes les bascules.

→ Ceci a pour conséquence de provoquer des états transitoires indésirables =  $n$  changement d'états →  $n-1$  états transitoires

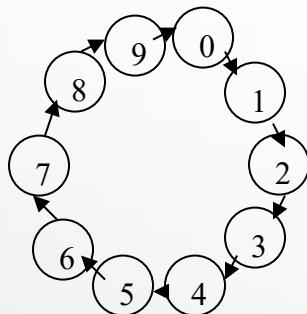
→ Exemple : supposons un temps de réponse  $t_r$  identique pour toutes les bascules. Considérons la chronologie du passage d'un compteur asynchrone 4 bits de 0111 à 1000.



Le compteur passe par les états transitoires 0110, 0100 et 0000 qui sont faux. Ceci est un inconvénient chaque fois que la sortie du compteur est exploitée par des organes rapides

# Logique séquentielle: Compteur asynchrone

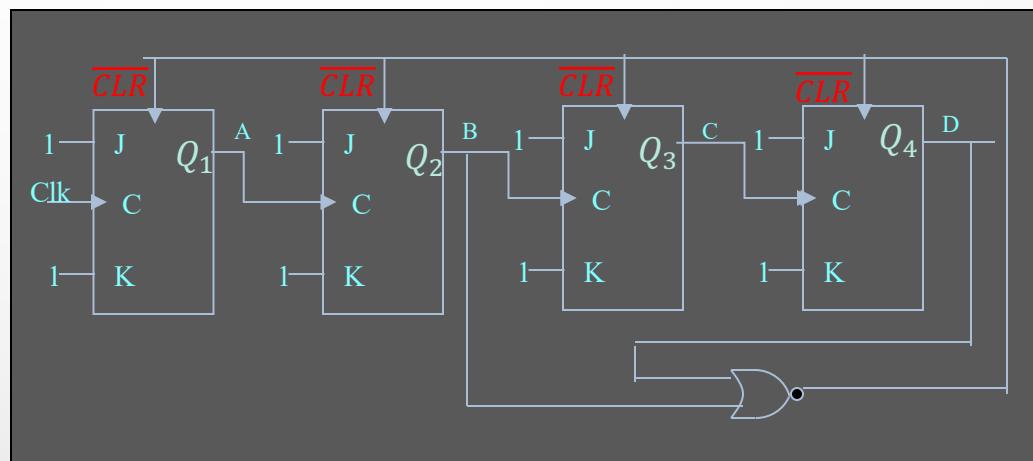
## Compteur modulo 10



Rebouclage asynchrone

Succession des états identiques (Modulo 16)  $\rightarrow$  9

Modulo 16	Modulo 10
0000	0000
0001	0001
.....	.....
1000	1000
1001	1001
1010	0000
1011	0001



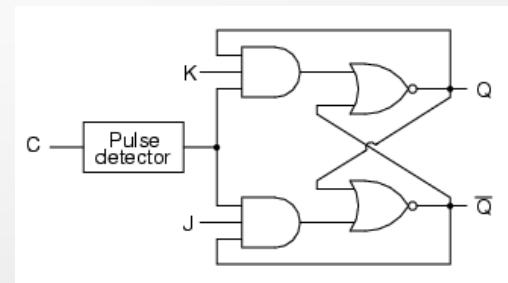
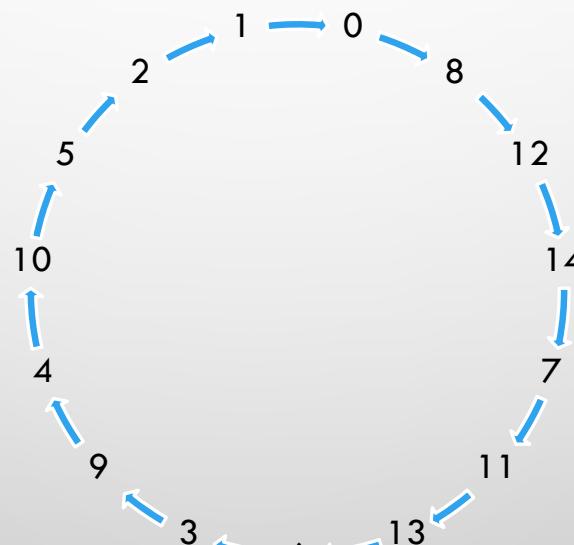
$D=1$  et  $B=1$  alors réinitialiser les bascules à 0  
 $\Rightarrow$  Clear

# Logique séquentielle

## Compteur synchrone

- bascules JK : séquences  $\{0,8,12,14,7,11,13,6,3,9,4,10,5,2,1,0\}$

15 états : 4bits, 4 bascules de sortie A,B,C,D



# Logique séquentielle

## Compteur synchrone

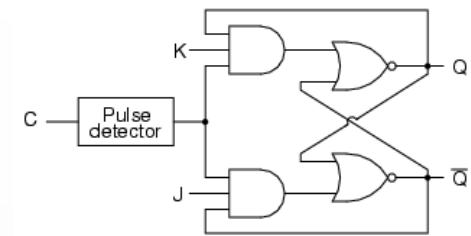
- bascules JK : séquences {0,8,12,14,7,11,13,6,3,9,4,10,5,2,1,0}

15 états : 4bits, 4 bascules de sortie A,B,C,D

$Q_{n-1} \rightarrow Q_n$	J K
0 → 0	0 X
0 → 1	1 X
1 → 1	X 0
1 → 0	X 1

- Automates des états
- Matrice de transition

N	Sorties ABCD	Entrées							
		J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>	J <sub>C</sub>	K <sub>C</sub>	J <sub>D</sub>	K <sub>D</sub>
0	0 0 0 0	1 X	0 X	0 X	0 X				
8	1 0 0 0	X 0	1 X	0 X	0 X				
12	1 1 0 0	X 0	X 0	1 X	0 X				
14	1 1 1 0	X 1	X 0	X 0	1 X				
7	0 1 1 1	1 X	X 1	X 0	0 X				
11	1 0 1 1	X 0	1 X	X 1	0 X				
13	1 1 0 1	X 1	X 0	1 X	X 1				
6	0 1 1 0	0 X	X 1	X 0	1 X				
3	0 0 1 1	1 X	0 X	X 1	X 0				
9	1 0 0 1	X 1	1 X	0 X	X 0				
4	0 1 0 1	1 X	X 1	1 X	X 1				
10	1 0 1 0	X 1	1 X	X 0	1 X				
5	0 1 0 1	0 X	X 1	1 X	X 0				
2	0 0 1 0	0 X	0 X	X 1	1 X				
1	0 0 0 1	0 X	0 X	0 X	X 1				
0	0 0 0 0								



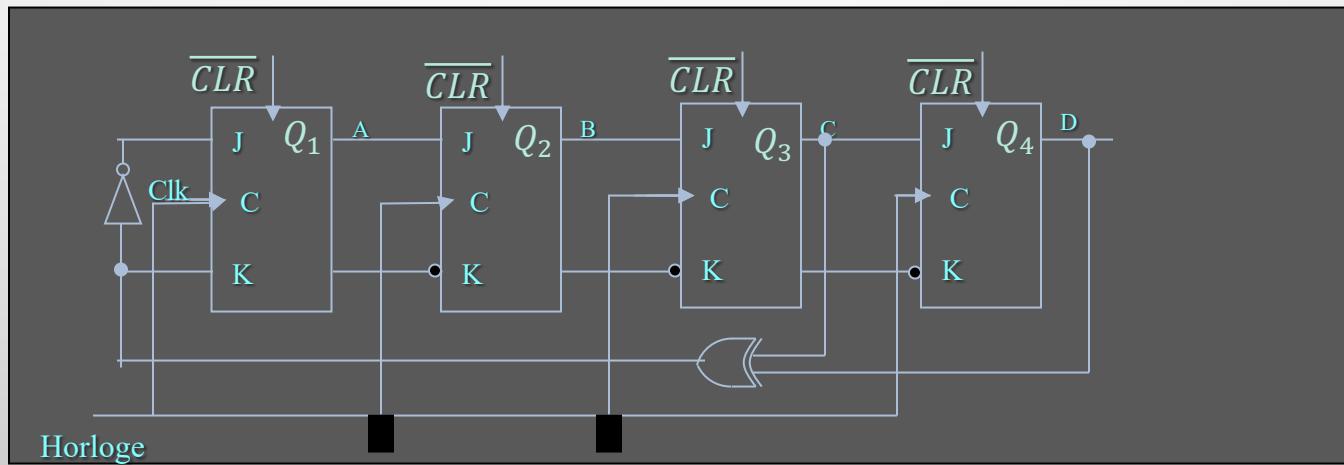
→ 8 tableaux de Karnaugh de sortie  
 $J_A, K_A, J_B, K_B, J_C, K_C, J_D, K_D$   
d'entrées A, B, C, D

# Logique séquentielle : Compteur synchrone

- bascules JK : séquences {0,8,12,14,7,11,13,6,3,9,4,10,5,2,1,0}

15 états : 4bits, 4 bascules de sortie A,B,C,D

$$\begin{array}{llll} J_A = \overline{C \oplus D} & K_A = C \oplus D & J_B = A & K_B = \overline{A} \\ & & J_C = B & K_C = \overline{B} \\ J_D = C & K_D = \overline{C} & & \end{array}$$

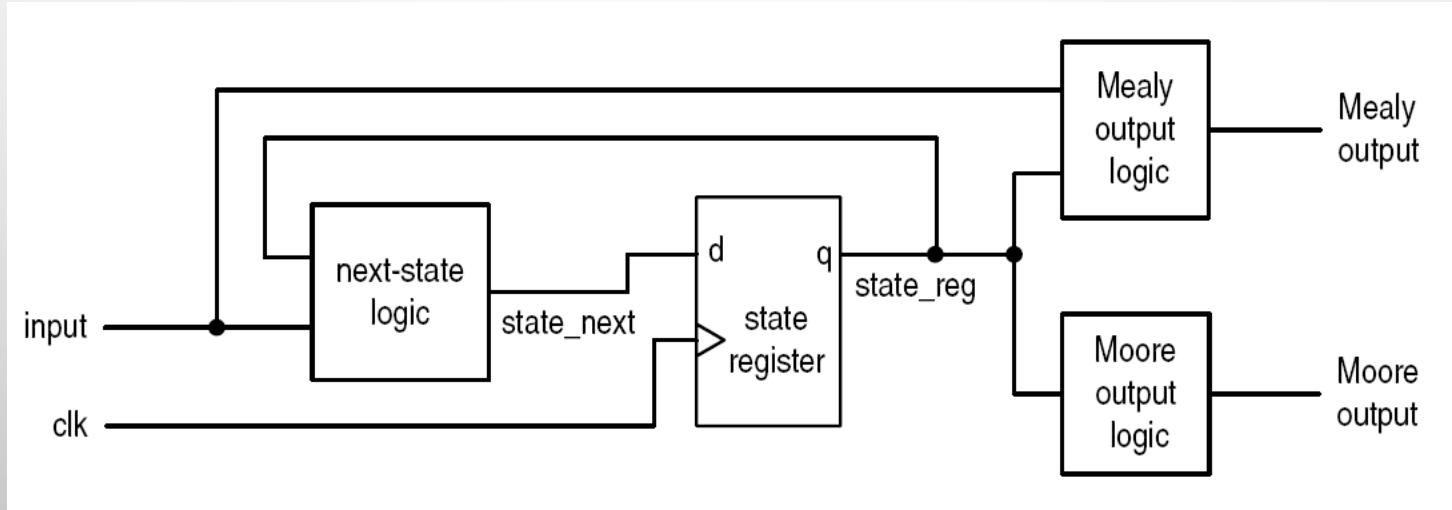


# Conception des systèmes numériques

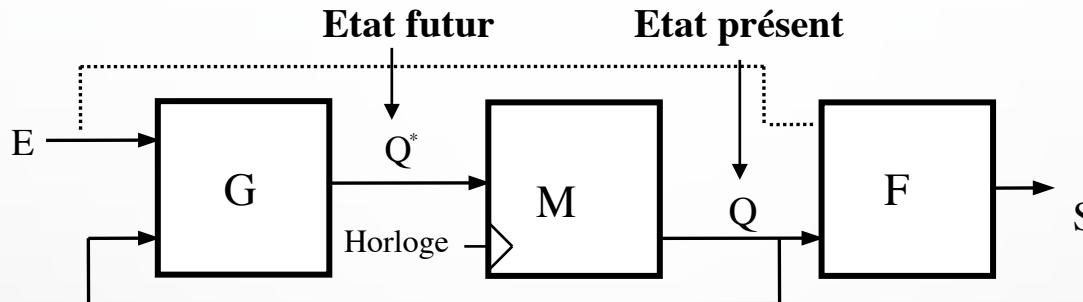
## 5. Machine à états finis

# Machines à états finis (Finite State Machines)

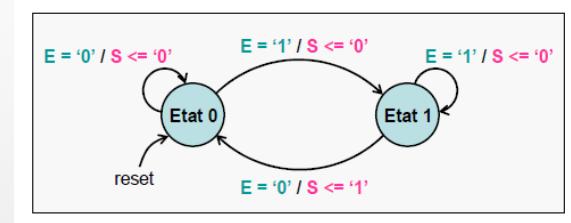
- Pour la réalisation de contrôleur dans un circuit complexe
- Deux types de machines : Mealy et Moore



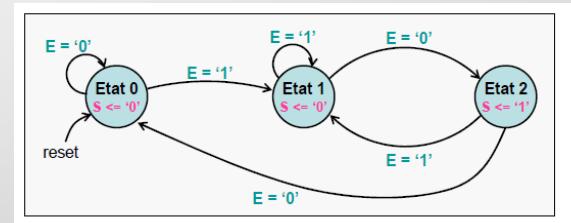
# Modèles Moore ou Mealy



Si les entrées (E) interviennent également dans l'état des sorties (S) (liaison pointillée), cette structure correspond au modèle de **Mealy**.



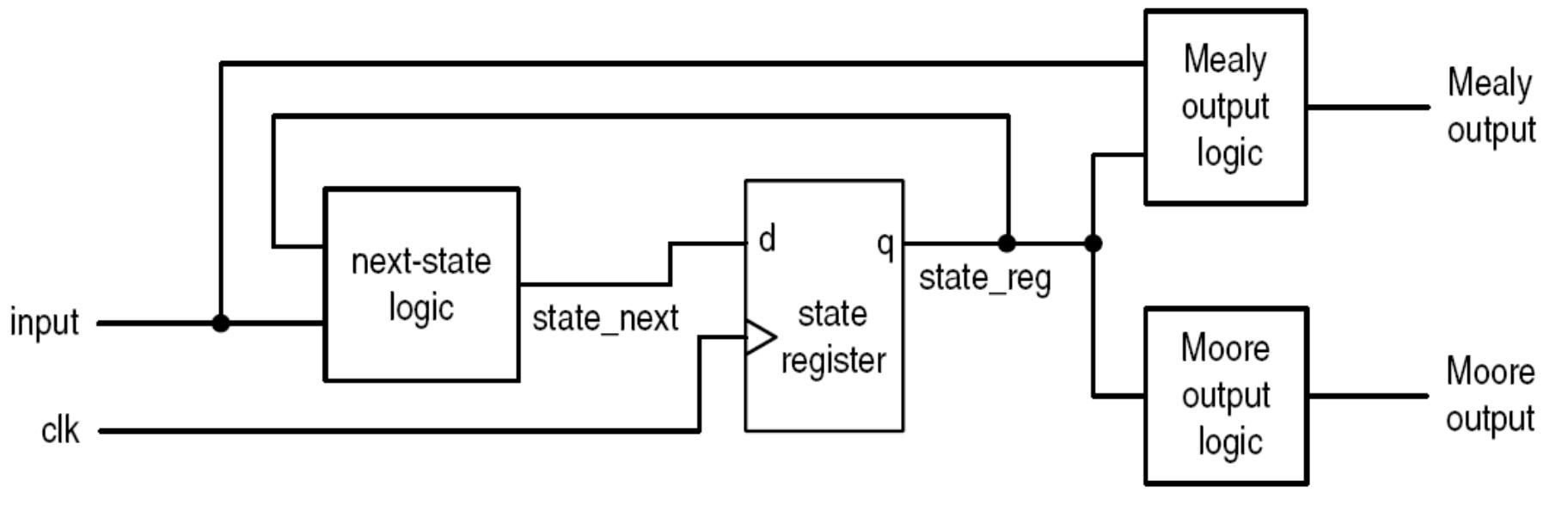
Lorsque les sorties (S) sont uniquement fonctions de l'état (Q) du système, cette structure implémente une **machine d'état** dite de **Moore**.



# Moore ou Mealy

- Machine de Mealy :
  - utilise en général moins d'états
  - plus rapide
  - transparente aux signaux transitoires (glitches)
- Laquelle des deux est meilleure ?
- La réponse dépend du type de signal de contrôle à réaliser
- Un signal sensible aux fronts montant ou descendant (edge sensitive)
  - Exemple : le signal d'activation d'un compteur
  - Les deux peuvent être utilisées mais la machine de Mealy est plus rapide
- Un signal sensible au niveau (level sensitive)
  - Exemple : un signal d'écriture d'une SRAM
  - Pour ce cas de figure, à préférer une machine de Moore

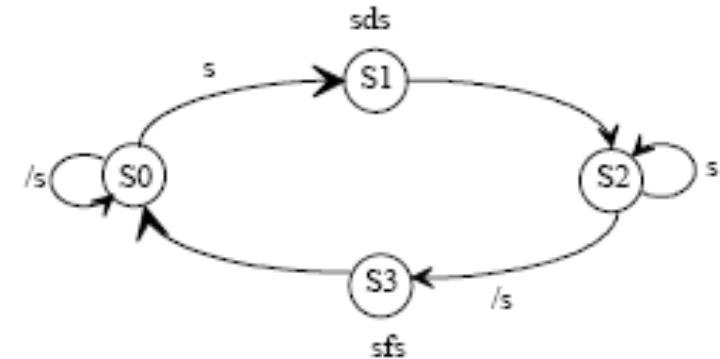
# Machines à états finis en VHDL



```

library ieee;
use ieee.std_logic_1164.all;
entity mooree is port(
    clk, reset : in std_logic;
    s : in std_logic;
    sds, sfs : out std_logic);
end mooree ;
architecture archmooree of mooree is
    type states is (s0, s1, s2, s3);
    signal state : states := s0;
begin
    process(reset, clk)
    begin
        if reset='1' then state <= s0;
        elsif (clk'event and clk= '1') then
            case state is
                when s0 => if s='1' then state <= s1;
                            else state <= s0; end if;
                when s1 => state <= s2;
                when s2 => if s='0' then state <= s3;
                            else state <= s2; end if;
                when s3 => state <= s0;
            end case;
        end if;
    end process;
    sds <= '1' when (state=s1) else '0';
    sfs <= '1' when (state=s3) else '0';
end archmooree;

```

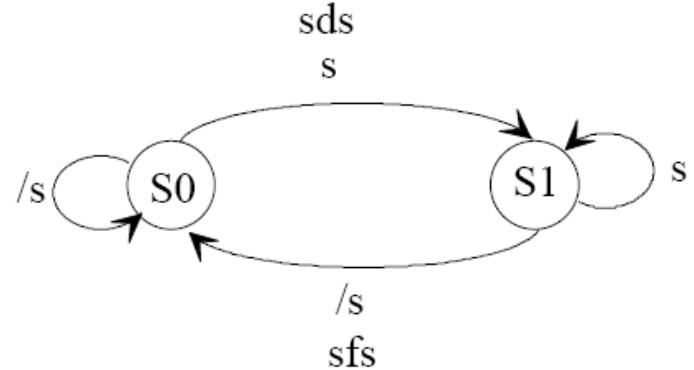


## Exemple Moore

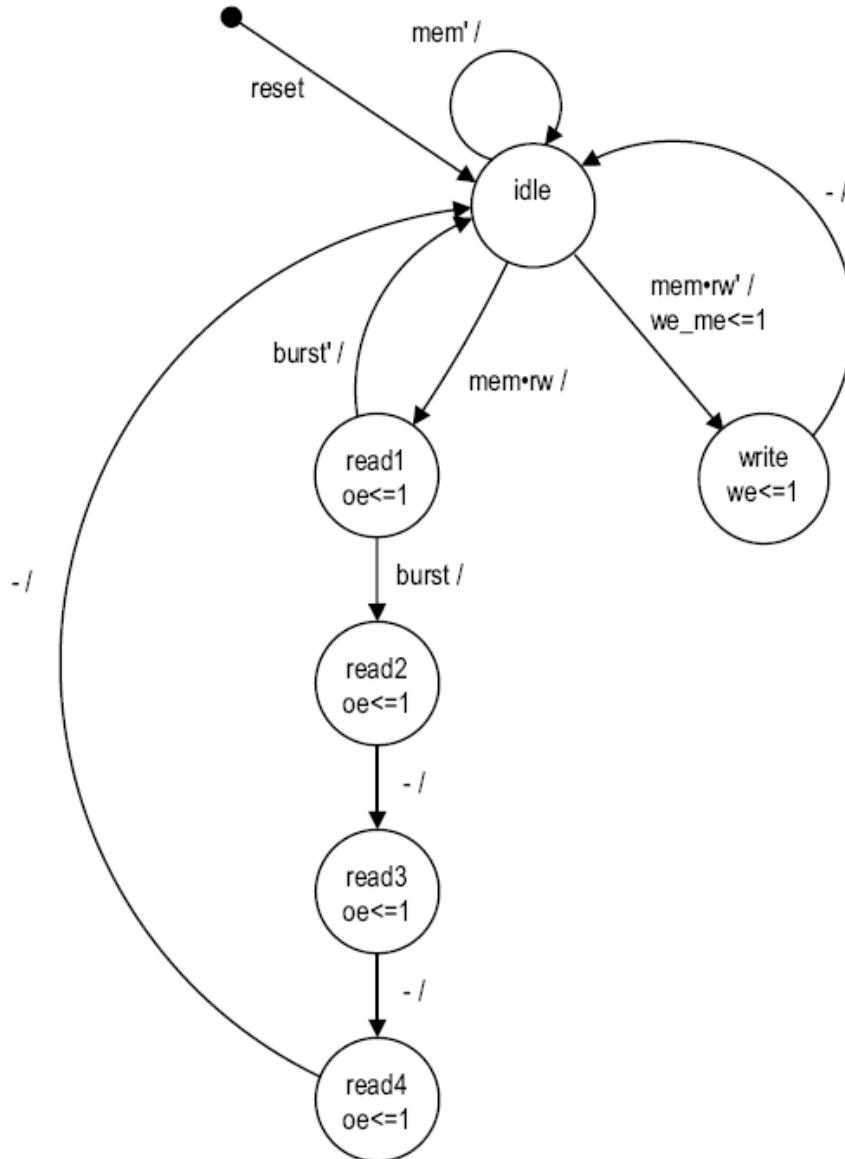
```

library ieee;
use ieee.std_logic_1164.all;
entity mealye is port(
    clk, reset : in std_logic;
    s : in std_logic;
    sds, sfs : out std_logic);
end mooree ;
architecture archmealye of mealye is
    type states is (s0, s1);
    signal state : states := s0;
begin
    process(reset, clk)
    begin
        if reset='1' then state <= s0;
        elsif (clk'event and clk= '1') then
            case state is
                when s0 => if s='1' then state <= s1;
                                else state <= s0; end if;
                when s1 => if s='0' then state <= s0;
                                else state <= s1; end if;
            end case;
        end if;
    end process;
    sds <= '1' when (state=s0 and s='1') else '0';
    sfs <= '1' when (state=s1 and s='0') else '0';
end archmealye;

```



## Exemple Mealy



```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is port(
    clk, reset: in std_logic;
    mem, rw, burst: in std_logic;
    oe, we, we_me: out std_logic);
end mem_ctrl ;
architecture mult_seg_arch of mem_ctrl is
    type mc_state_type is (idle, read1, read2, read3, read4, write);
    signal state_reg, state_next: mc_state_type;
begin
    -- state register
    process(clk,reset)
    begin
        if (reset='1') then
            state_reg <= idle;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;
    -- next-state logic
```

```
-- next-state logic
process(state_reg,mem,rw,burst)
begin
    case state_reg is
        when idle =>
            if mem='1' then
                if rw='1' then
                    state_next <= read1;
                else
                    state_next <= write;
                end if;
            else
                state_next <= idle;
            end if;
        when write => state_next <= idle;
        when read1 =>
            if (burst='1') then
                state_next <= read2;
            else
                state_next <= idle;
            end if;
        when read2 => state_next <= read3;
        when read3 => state_next <= read4;
        when read4 => state_next <= idle;
    end case;
end process;
```

```
-- Moore output logic
process(state_reg)
begin
    we <= '0'; -- default value
    oe <= '0'; -- default value
    case state_reg is
        when idle =>
        when write =>
            we <= '1';
        when read1 =>
            oe <= '1';
        when read2 =>
            oe <= '1';
        when read3 =>
            oe <= '1';
        when read4 =>
            oe <= '1';
    end case;
end process;
```

-- Mealy output logic

```
process(state_reg,mem,rw)
```

```
begin
```

```
    we_me <= '0'; -- default value
```

```
    case state_reg is
```

```
        when idle =>
```

```
            if (mem='1') and (rw='0') then
```

```
                we_me <= '1';
```

```
        end if;
```

```
        when write =>
```

```
        when read1 =>
```

```
        when read2 =>
```

```
        when read3 =>
```

```
        when read4 =>
```

```
    end case;
```

```
end process;
```

```
we_me <= '1' when ((state_reg=idle) and (mem='1') and (rw='0')) else '0';
```

```
end mult_seg_arch;
```