

7. CIRCUITS NUMÉRIQUES LOGIQUE

Initiation à l'électronique

JEAN-MICHEL SALLESE

CIRCUITS NUMÉRIQUES

Traitement numérique des signaux

Système binaire et logique binaire

Opérateurs logiques et algèbre de Boole

Systèmes logiques combinatoires

Élément de mémoire et bascule

Systèmes logiques séquentiels

TRAITEMENTS NUMÉRIQUES DES SIGNAUX

MONDE ANALOGIQUE

GRANDEURS À VARIATION CONTINUE

Un **continuum** de valeurs possibles
dans un intervalle donné

Exemples:
force, position, température
tension, courant, puissance

Traitement analogique

*Amplifier, filtrer, déformer,
additionner, soustraire*

MONDE NUMÉRIQUE

GRANDEURS À VALEURS DISCRÈTES

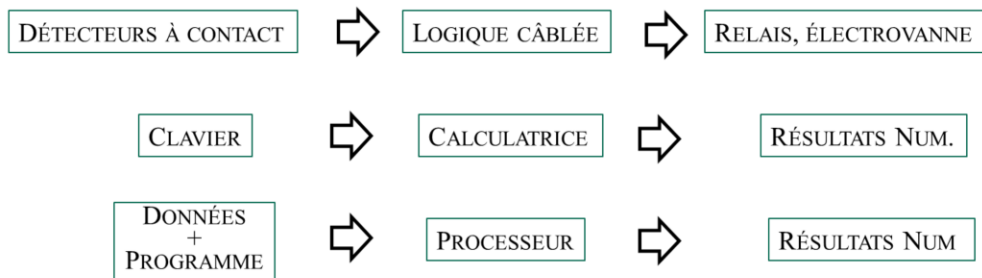
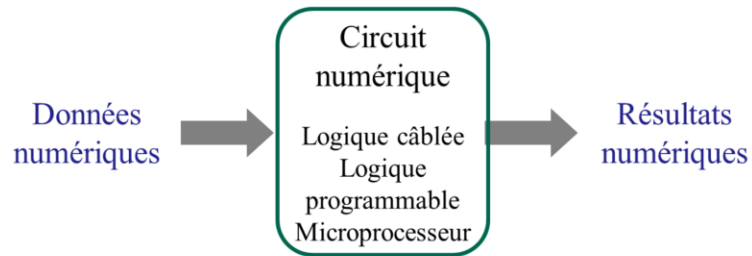
Un **nombre fini** de valeurs possibles
pouvant être codées avec des
symboles

Exemples:
nombre d'objets
états discrets d'un ensemble

Traitement numérique

Calculs, opérations logiques

TRAITEMENT NUMÉRIQUE DE GRANDEURS NUMÉRIQUES

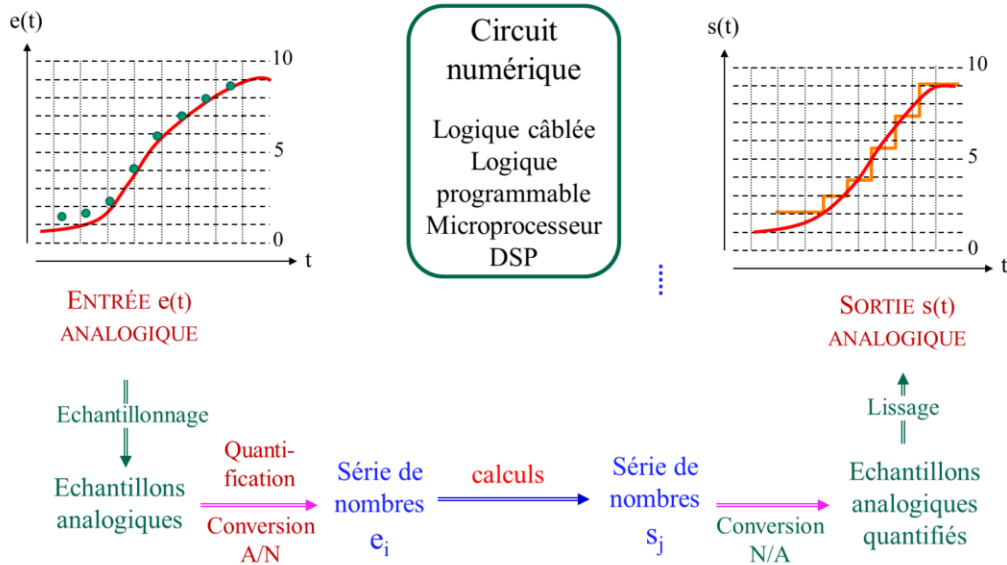


De nombreux systèmes sont entièrement numériques.

Leurs données sont d'origine numériques, et les résultats sont directement utilisés sous cette forme numérique.

Ces résultats peuvent également servir de données numériques.

TRAITEMENT NUMÉRIQUE DE GRANDEURS ANALOGIQUES



Initiation à l'électronique - Chapitre 7 – Logique et circuits numériques - page 6

L'électronique numérique est aussi très largement utilisée dans le traitement de grandeurs analogiques.

Grace à la rapidité d'exécution des calcul par les processeurs, elle peut remplacer une partie des circuits analogiques traditionnels.

Pour cela, il faut pouvoir convertir des variables analogiques continues en variables numériques discrète grâce à des convertisseurs A/N, analogique-numérique.

De même, pour 'revenir' aux signaux analogiques après que les calculs aient été effectué, il faut pouvoir convertir des signaux numériques en grandeurs analogiques (tensions ou courants).

C'est ce que réalisent les convertisseurs N/A, numérique-analogiques.

AVANTAGES ET INCONVÉNIENTS DU TRAITEMENT NUMÉRIQUE

Avantages

Insensibilité au bruit: tant que l'on peut identifier les chiffres

Précision: limitée uniquement par le nombre de chiffres calculés

Souplesse: toute fonction que l'on peut mettre en équation ou tabuler

Inconvénients

Conversion A/N et N/A: limites en termes de précision et de vitesse

Complexité: nombre élevé de composants et de connections

Temps de traitement: un calcul, même simple, peut nécessiter beaucoup d'opérations, donc un temps important

Ces inconvénients sont minimisés par l'augmentation des performances (densité, vitesse, précision) des circuits intégrés

Initiation à l'électronique - Chapitre 7 – Logique et circuits numériques - page 7

L'hégémonie des circuits numériques est due aux progrès constants des techniques d'intégration qui permettent de réaliser des circuits intégrés de plus en plus complexes et rapides avec un coût par fonction qui ne cesse de diminuer.

SYTÈME BINAIRE – ALGÈBRE DE BOOLE

SYSTEME BINAIRE

Chiffre (Bit) : $\mathbf{a}_i = [1, 0]$

Représentation d'un nombre en base 2 : $a_{n-1} a_{n-2} \dots a_2 a_1 a_0$

$\uparrow$ MSB
 $\uparrow$ LSB

valeur : $A = \sum_{i=0}^{i=n-1} a_i \cdot 2^i$

Exemple

notation binaire : 10011

valeur décimale: $A = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
 $= 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$
 $= 19$

MSB = Most Significant Bit = le bit de poids le plus fort (2^{n-1})

LSB = Least Significant Bit = le bit de poids le plus faible (2^0)

LOGIQUE BINAIRE

Une variable binaire ne peut avoir que deux valeurs appelées aussi *Etats*

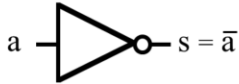
<i>Vrai</i> <i>1</i>	<i>5 Volts - High</i>	 <i>Close</i>
<i>Faux</i> <i>0</i>	<i>0 Volt - Low</i>	 <i>Open</i>
<i>VALEUR LOGIQUE</i>	<i>EQUIVALENT ÉLECTRIQUE</i>	<i>EQUIVALENT ÉLECTRIQUE</i>

A quelques rares exceptions près, les circuits électroniques numériques représentent le "0" logique par une tension quasi nulle par rapport à la masse (référence 0 V), et le "1" logique par une tension positive, généralement proche de la tension d'alimentation, par rapport à la masse (référence 0 V).

ALGÈBRE DE BOOLE

OPÉRATEURS LOGIQUES BINAIRES FONDAMENTAUX

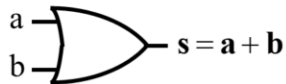
Négation (NOT)



a	s
0	1
1	0

"inverseur"

Fonction OU (OR)



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

Fonction ET (AND)



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

Fonction OU exclusif (XOR)



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

Négation, aussi appelée **inversion**, notée $\bar{}$ ou parfois /

La sortie est l'opposé (ou complément) de l'entrée.

OU, notée +

Il suffit que l'une des entrées soit vraie pour que la sortie soit vraie.

ET, notée $\cdot$ ou * ou encore &

Il est nécessaire que toutes les entrées soient vraies pour que la sortie soit vraie.

Il est important de ne pas confondre les notations + et $\cdot$ logiques avec l'addition et la multiplication des nombres, même si, en algèbre de Boole, on utilise parfois les termes "somme" et "produit" sous-entendu "logique" .

La table de vérité d'une fonction logique à n variables est un tableau listant le résultat correspondant à chacune des 2^n combinaisons possibles des n variables d'entrée.

RÈGLES DE L'ALGÈBRE DE BOOLE

$$\begin{array}{lllll} \overline{\overline{a}} = a & a \cdot 0 = 0 & a + 0 = a & a \cdot 1 = a & a + 1 = 1 \\ & a \cdot a = a & a + a = a & a \cdot \overline{a} = 0 & a + \overline{a} = 1 \end{array}$$

Commutativité	$a \cdot b = b \cdot a$	$a + b = b + a$
Distributivité	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	$a + (b \cdot c) = (a + b) \cdot (a + c)$
Associativité	$a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c$	$a + (b + c) = (a + b) + c = a + b + c$
Consensus	$(a \cdot c) + (b \cdot \overline{c}) + (a \cdot b) = (a \cdot c) + (b \cdot \overline{c})$ $(a + c) \cdot (b + \overline{c}) \cdot (a + b) = (a + c) \cdot (b + \overline{c})$	
	$a + (a \cdot b) = a$	$a + (\overline{a} \cdot b) = a + b$
Loi De Morgan	$\overline{a \cdot b} = \overline{a} + \overline{b}$	$\overline{a + b} = \overline{a} \cdot \overline{b}$

Démonstration en établissant la table de vérité de chaque terme

Initiation à l'électronique - Chapitre 7 – Logique et circuits numériques - page 12

La loi de Morgan est utile pour simplifier (factoriser) l'architecture de circuits numériques, ce qui revient à réduire le nombre de portes logiques utilisées pour réaliser une fonction.

Démonstration du théorème de De Morgan

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

$$\overline{a+b} = \overline{a}\cdot\overline{b}$$

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

$$\overline{a}\cdot\overline{b} = \overline{a+b}$$

Démonstration du théorème de distributivité

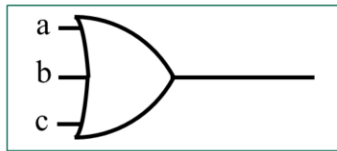
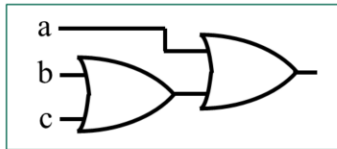
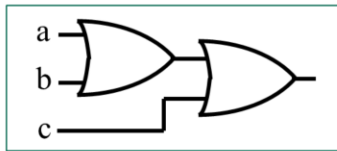
a	b	c	b+c	a·b	a·c	$a\cdot(b+c)$	$(a\cdot b)+(a\cdot c)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

$$a\cdot(b+c) = (a\cdot b)+(a\cdot c)$$

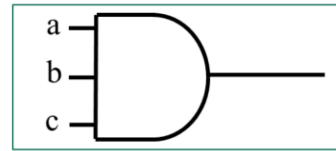
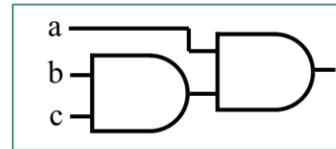
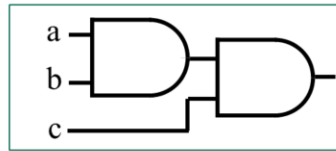
a	b	c	b·c	a+b	a+c	$a+(b\cdot c)$	$(a+b)\cdot(a+c)$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

$$a+(b\cdot c) = (a+b)\cdot(a+c)$$

COROLLAIRE DU THÉORÈME D'ASSOCIATIVITÉ



$$(a+b)+c = a+(b+c) = a+b+c$$



$$a(bc) = abc = (ab)c$$

Extensible à n entrées

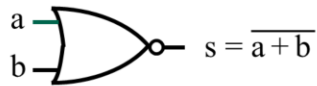
La définition des fonctions logiques OU et ET reste valable pour un nombre quelconque d'entrées.

Des réalisations physiques sous forme de circuits intégrés standards sont disponibles avec 2, 3, 4 et 8 entrées.

Les circuits logiques programmables offrent encore plus de souplesse.

OPÉRATEURS LOGIQUES UNIVERSELS

Fonction *NOT-OR* (NOR)



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

Fonction *NOT-AND* (NAND)



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

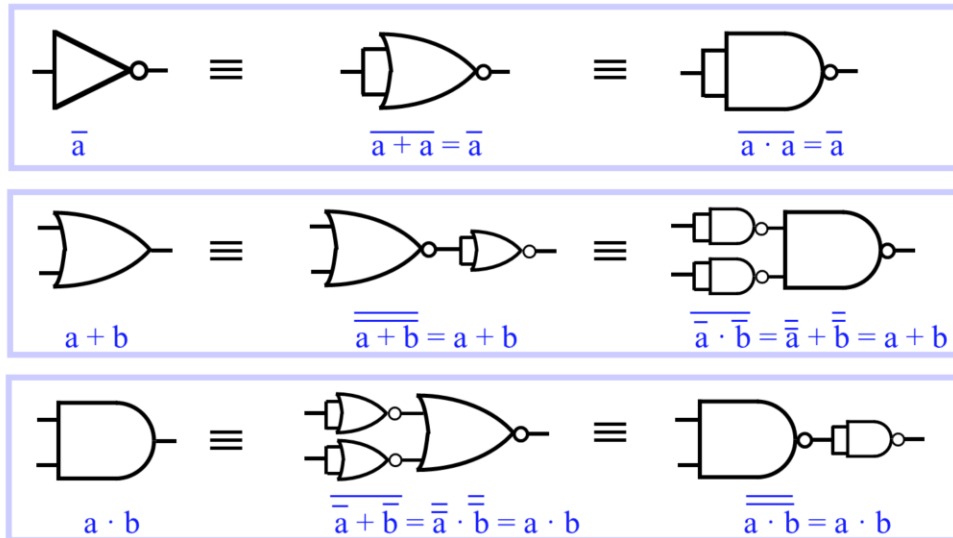
Le 'o' de sortie symbolise la négation

Ces opérateurs sont simples à réaliser avec des composants électroniques

En circuits intégrés standards du commerce, les portes NAND et NOR sont disponibles avec 2, 3, 4 et 8 entrées.

DÉMONSTRATION DE L'UNIVERSALITÉ DES FONCTIONS NAND ET NOR

Les trois fonctions de base peuvent être réalisées avec des NOR ou des NAND.



Avec uniquement des opérateurs NOR, respectivement NAND, on peut réaliser les trois fonctions de base, et donc toute fonction logique.

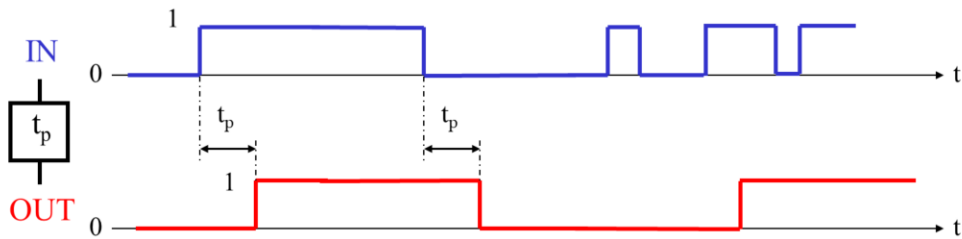
De même, on peut faire la fonction NOR avec uniquement des fonctions NAND, et des fonctions NAND avec uniquement des fonctions NOR.

DYNAMIQUE DES OPÉRATEURS LOGIQUES

Le temps de réaction d'un opérateur logique est modélisé par un élément de retard à la sortie de l'opérateur idéal (NOT, ET, OU, NAND, NOR).

Il introduit un temps de propagation t_p entre un changement d'état de sa sortie vis-à-vis de son entrée.

Un intervalle de temps inférieur à t_p n'aura aucun effet sur la sortie.



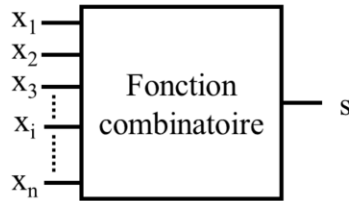
Ce comportement de l'élément de retard modélise le fait qu'un opérateur réel a une limite interne en terme de temps de réaction.

Lorsque les signaux changent rapidement d'état, par ex. en quelques centaines ou dizaines de nanosecondes, voire moins, le retard intrinsèque aux circuits numériques doit être pris en compte pour des questions de synchronisation.

Les simulateurs électriques intègre cette donnée.

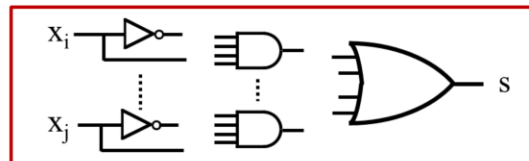
LOGIQUE COMBINATOIRE

FONCTION LOGIQUE COMBINATOIRE QUELCONQUE



A chacune des 2^n combinaisons des entrées x_i correspond une valeur de sortie

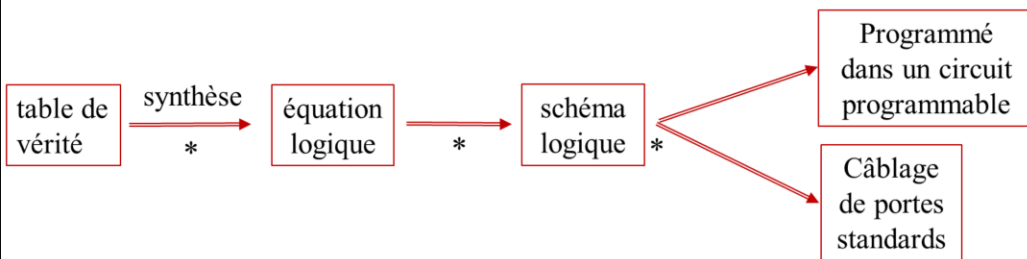
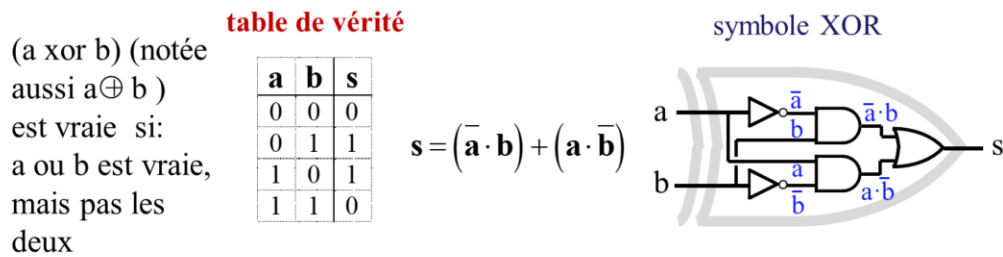
Toute fonction combinatoire peut être décomposée en une "*somme*" (OR) de "*produits*" (AND) des entrées ou de leurs inverses (NOT).



Dans un système dit 'combinatoire, une suite donnée de variables d'entrée donnera toujours le même résultat en sortie.

Le signal de sortie ne dépend donc pas des signaux passés.

EX. DE FONCTION COMBINATOIRE : FONCTION "OU EXCLUSIF" (XOR)



(* réalisé avec des outils CAO)

Bien que dans les cas simples, la synthèse et la réalisation peuvent se faire manuellement, on utilise des programmes spécialisés de synthèse et de simulation.

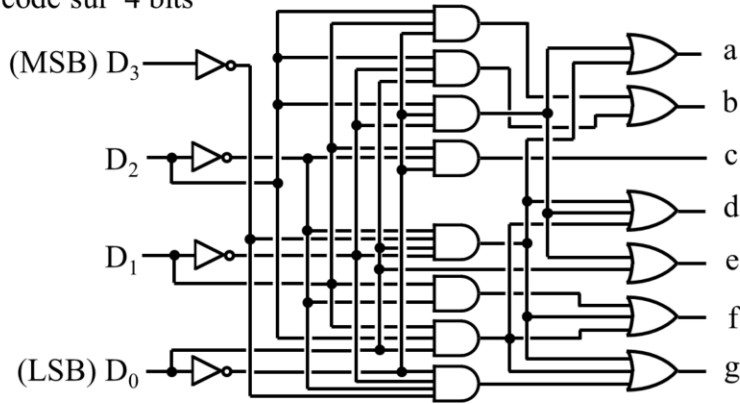
Les réalisations en logique câblée avec des circuits standards ne sont possibles que pour des cas relativement simples. C'est le cas de la fonction XOR qui est si utile qu'elle est considérée comme une fonction de base, avec un symbole spécifique, et disponible en circuit standard.

Actuellement les fonctions logiques sont généralement réalisées avec des circuits programmables tels que des FPGA (Field Programmable Gate Array).

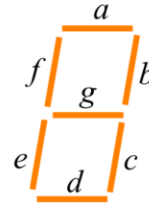
Une autre solution serait d'utiliser un microcontrôleur et d'y implanter un programme qui effectue l'opération logique. Tous les microcontrôleurs ont dans leur jeu d'instructions les fonctions AND, OR, NOT et XOR.

EXEMPLE 1 : DÉCODEUR BCD - 7 SEGMENTS

Chiffre décimal
codé sur 4 bits



Affichage
décimal à 7
segments



Le segment x est visible lorsque la sortie logique $x = 0$

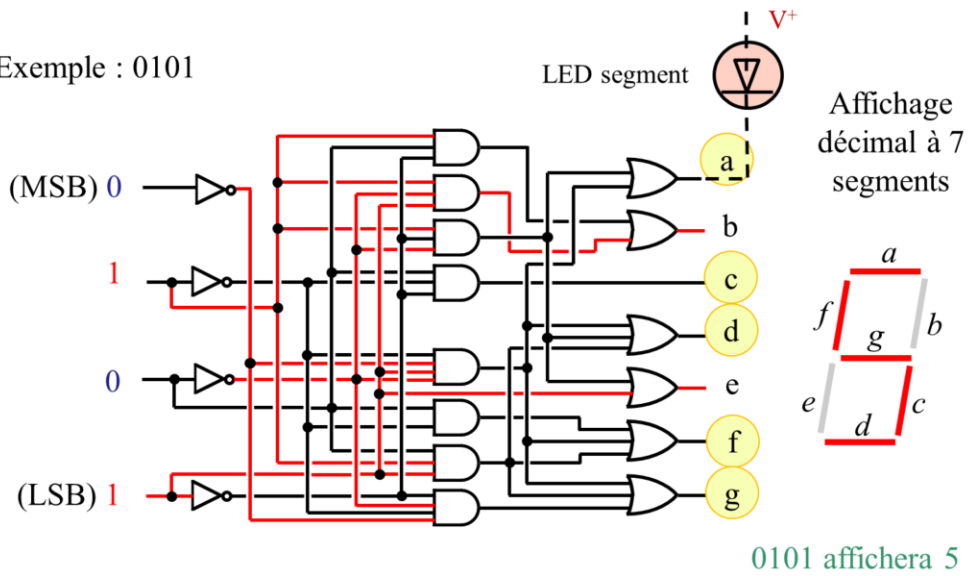
En code BCD les chiffres décimaux 0 à 9 sont codés en binaire. Ce code est souvent utilisé lorsqu'un nombre doit être affiché sous une forme décimale.

L'exemple ci-dessus donne le schéma d'un décodeur qui va faire apparaître les segments représentant le chiffre qui correspond au code binaire appliqué à l'entrée.

Chaque sortie de **a** à **g** est une fonction combinatoire des entrées D_0 à D_3

EXEMPLE 1 : DÉCODEUR BCD - 7 SEGMENTS

Exemple : 0101



Attention : Le segment x est visible lorsque la sortie logique $x=0$

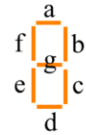
Exemple du décodage du nombre binaire 101, qui est 5 en base 10.

En rouge sont les 'signaux' au potentiel V^+ , et en noir 0V.

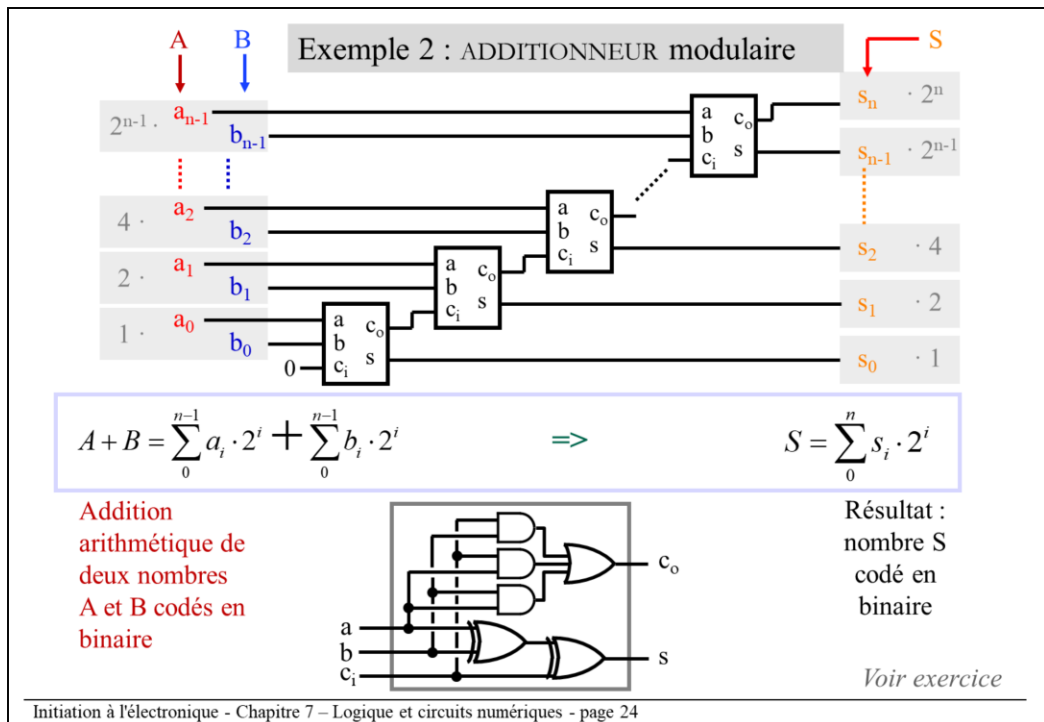
Attention: ici les LED s'allument si on applique 0V aux sorties du décodeur.

EXEMPLE 1 : DÉCODEUR BCD - 7 SEGMENTS

D3	D2	D1	D0	a	b	c	d	e	f	g	display
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	0	1	0	0	0	0	0	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	0	1	0	0	9



! : Le segment x apparaît lorsque la sortie logique correspondante est à 0



Pour faire l'addition de deux nombres en décimal:

retenue	1 1
A	1 6 5 3
B	2 5 7
S	1 9 1 0

pour chaque puissance de dix (unités, dizaines, centaines, ...) on additionne les deux chiffres et l'éventuelle retenue de l'addition précédente, ce qui donne le chiffre de la somme et une éventuelle retenue pour l'addition suivante.

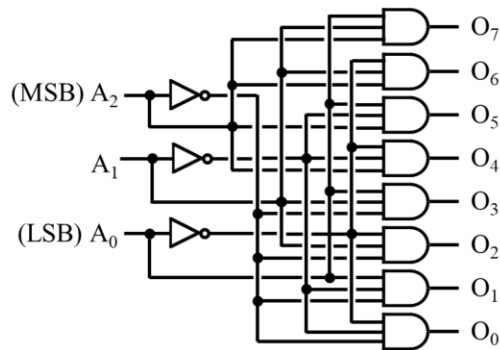
On peut faire la même chose avec deux nombres codés en binaire:

retenue	1 1 1
A	1 0 0 1 1 0
B	1 1 1 1
S	1 1 0 1 0 1

EXEMPLE 3 : DÉCODEUR 3 - 8

Adresse
de 3 bits

Sélection de 1 sortie
parmi 8



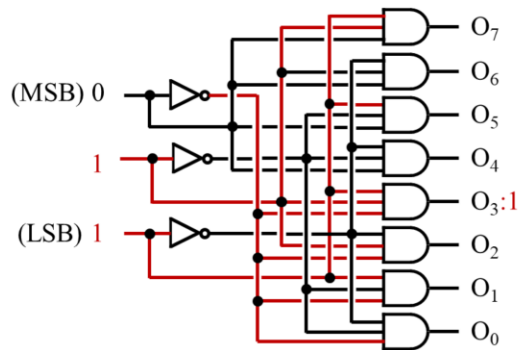
A ₂	A ₁	A ₀	O ₀	O ₁	O ₂	O ₃	O ₄	O ₅	O ₆	O ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Ce décodeur active une seule des sorties numérotées de 0 à 7 en fonction de l'adresse codée en binaire à l'entrée via A₀ A₁ A₂.

EXEMPLE 3 : DÉCODEUR 3 - 8

011
(3 en base 10)

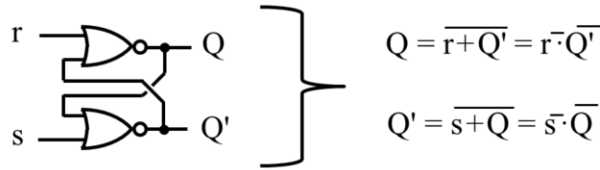
Sortie Q_3



A ₂	A ₁	A ₀	O ₀	O ₁	O ₂	O ₃	O ₄	O ₅	O ₆	O ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

ELEMENT DE MÉMOIRE ET BASCULE

ELÉMENT DE MÉMOIRE, BASCULE BISTABLE SR

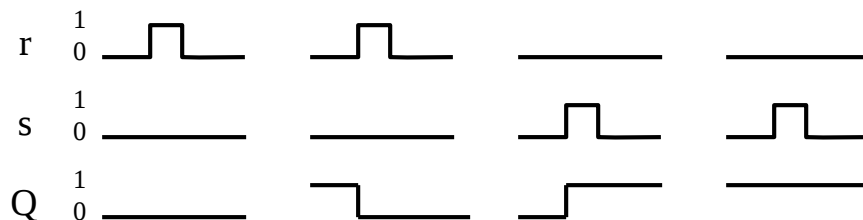


$s = 0$ et $r = 1$	$Q = 0$	$Q' = \overline{Q} = 1$	Q est mis à 0 $\Leftrightarrow$ "reset"
$s = 1$ et $r = 0$	$Q' = 0$	$Q = \overline{Q'} = 1$	Q est mis à 1 $\Leftrightarrow$ "set"
$s = 0$ et $r = 0$	$Q' = \overline{Q}$	$Q = \overline{Q'} = Q$	état stable, qu'il soit 1 ou 0
$s = 1$ et $r = 1$	$Q = 0$	$Q' = 0 \neq \overline{Q}$	état à éviter !

La bascule bistable, mise dans un état donné par activation de l'une des deux entrées set ou (exclusif) reset, conserve ensuite cet état tant que les entrées restent inactives.

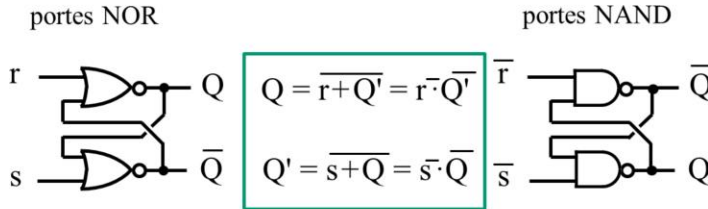
L'état d'entrée $s = r = 1$ est en général interdit car, d'une part les deux sorties Q et Q' ne sont alors plus complémentaires, d'autre part et surtout si s et r reviennent à 0 simultanément il est impossible de prévoir l'état que prendra la sortie Q !

Si l'on respecte l'interdiction de l'état $s = r = 1$, on peut rencontrer les quatre cas suivants:



Ce système est dit séquentiel et son comportement est fondamentalement différent de celui d'un système combinatoire puisque **la même combinaison des entrées s et r (dans ce cas 0 et 0) peut donner des résultats différents en fonction des événements passés.**

RÉALISATION DE L'ÉLÉMENT DE MÉMOIRE SR



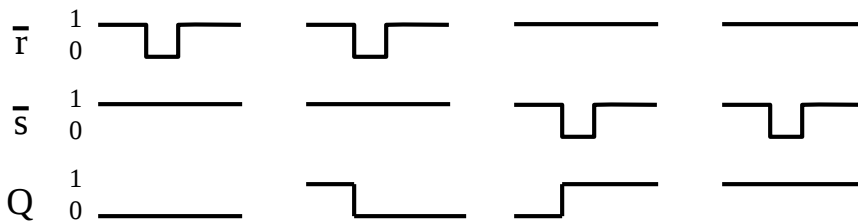
On passe d'un schéma à l'autre en appliquant le théorème de De Morgan

L'élément de mémoire réalisé avec des portes NAND est mis dans un état donné par un 0 sur l'une des entrées.

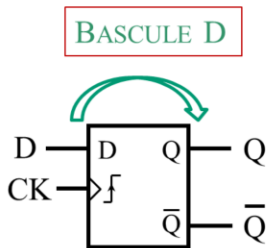
Les deux entrées ne doivent jamais être à 0 simultanément.

ELLE CONSERVE SON ÉTAT TANT QUE LES ENTRÉES RESTENT À 1.

L'élément de mémoire réalisé avec des portes NAND est mis dans un état donné par un 0 sur l'une ou (exclusif) l'autre des entrées, elle conserve son état inchangé tant que les entrées restent inactives à 1. Les deux entrées ne doivent jamais être à 0 simultanément.



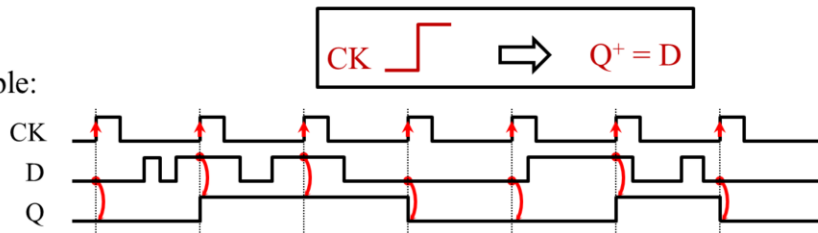
D-TYPE EDGE TRIGGERED FLIP FLOP BASCULE D (COMMANDÉE PAR UNE HORLOGE)



Au **front montant** du signal d'horloge CK, l'état de l'entrée D est envoyé sur la sortie Q.
Cette sortie garde sa valeur **jusqu'au prochain front montant de CK**, même si l'entrée D change entre temps.

L'état futur Q^+ est celui de l'entrée D au moment du front montant de CK.

Exemple:

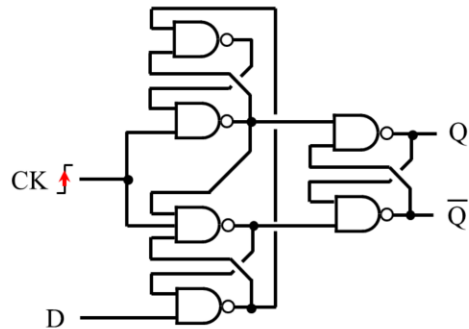


Seule la bascule D sera étudiée ici, car c'est la plus répandue, en particulier dans les circuits logiques programmables.

Mais il existe d'autres types de "Edge Triggered Flip Flop", en particulier la bascule JK.

Toute bascule d'un type donné peut être transformée en une bascule d'un autre type par ajout de logique combinatoire.

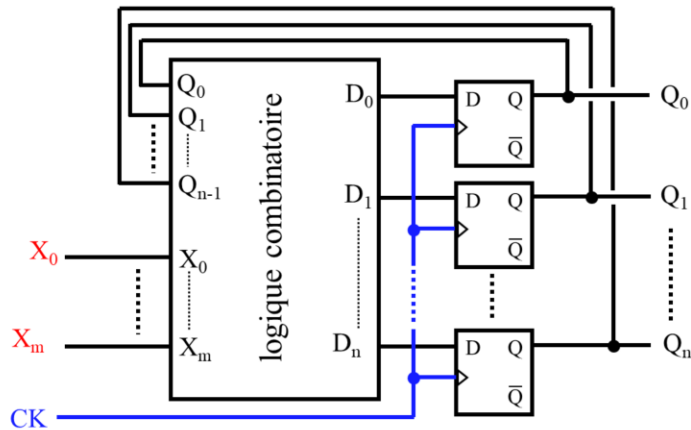
EXEMPLE DE RÉALISATION DE LA BASCULE D



Ce schéma est donné à titre d'information pour montrer qu'une fonction aussi simple nécessite déjà 6 portes élémentaires, ce qui représente déjà une vingtaine de transistors en logique CMOS .

SYSTÈMES LOGIQUES SÉQUENTIELS SYNCHRONES

SYSTÈMES LOGIQUES SÉQUENTIELS SYNCHRONES

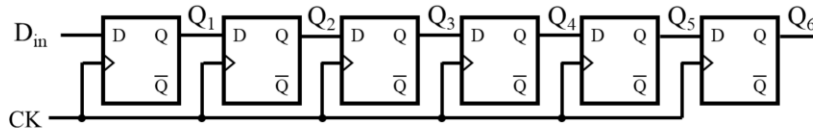


Juste après le front montant du signal d'horloge CK, chaque bascule prendra un nouvel état (dit futur) $Q_i^+ = D_i$ qui est une fonction combinatoire des états présents de toutes les bascules Q_0 à Q_n **et de toutes les entrées X_0 à X_m** à l'instant du front d'horloge.

Le système est dit synchrone car les bascules ont toutes le même signal d'horloge et ne peuvent donc changer d'état qu'en présence du front de cette horloge.

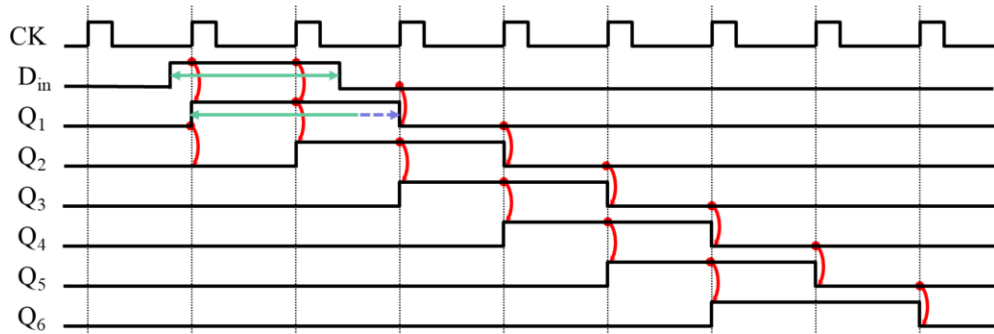
Par contre, le nouvel état de sortie $Q_0^+ \dots Q_n^+$ juste après le front montant de l'horloge sera fonction de l'état actuel $Q_0 \dots Q_n$ et aussi des entrées $X_0 \dots X_m$.

EXEMPLE 1 : REGISTRE À DÉCALAGE (SHIFT REGISTER)



$$D_i = Q_{i-1} \Rightarrow Q_i^+ = Q_{i-1}$$

A chaque impulsion d'horloge, une bascule i prend l'état de la précédente $i-1$.



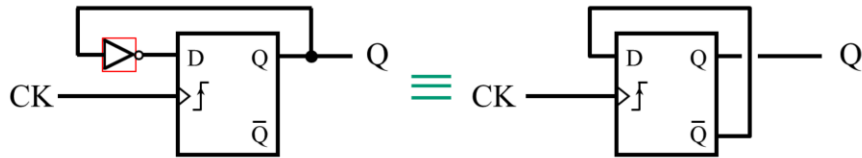
Il s'agit d'un circuit synchrone.

En reliant les bascules comme ci-dessus, et en appliquant le même signal d'horloge à toutes, on décale la valeur des entrée D à chaque signal d'horloge.

Plus précisément, la première bascule échantillonne l'entrée D_{in} aux fronts montants de l'horloge

Cette information est ensuite transférée d'une bascule dans la suivante à chaque front montant de l'horloge.

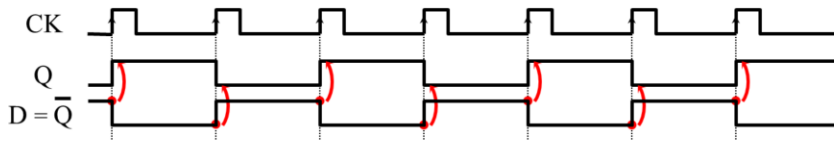
EXEMPLE 2 : COMPTEUR / DIVISEUR PAR 2



$$D = \bar{Q} \Rightarrow Q^+ = \bar{Q}$$

A chaque signal d'horloge la bascule change d'état.

Elle compte donc par 2.



La fréquence du signal de sortie Q est la moitié de celle du signal d'horloge en entrée, d'où l'appellation diviseur (de fréquence) par deux.

Le bouclage de la sortie NOT Q vers l'entrée D revient à diviser par 2 le nombre d'impulsions de l'horloge CK

EXEMPLE 3 : COMPTEUR SYNCHRONES PAR 5

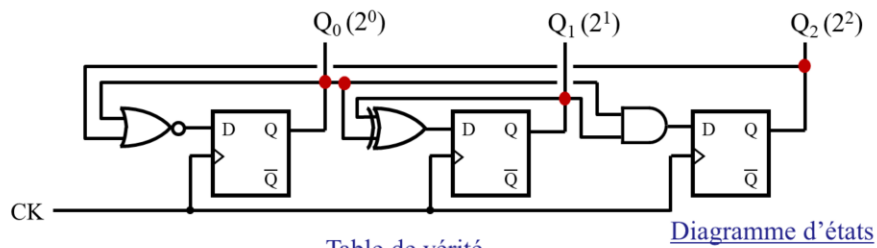


Table de vérité

Diagramme d'états

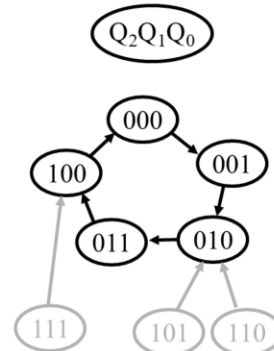
Equations

$$Q_2^+ = D_2 = Q_0 \cdot Q_1$$

$$Q_1^+ = D_1 = Q_0 \text{ XOR } Q_1$$

$$Q_0^+ = D_0 = \overline{Q_0 + Q_2}$$

Q ₂	Q ₁	Q ₀	Q ₂ ⁺ = D ₂	Q ₁ ⁺ = D ₁	Q ₀ ⁺ = D ₀
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0



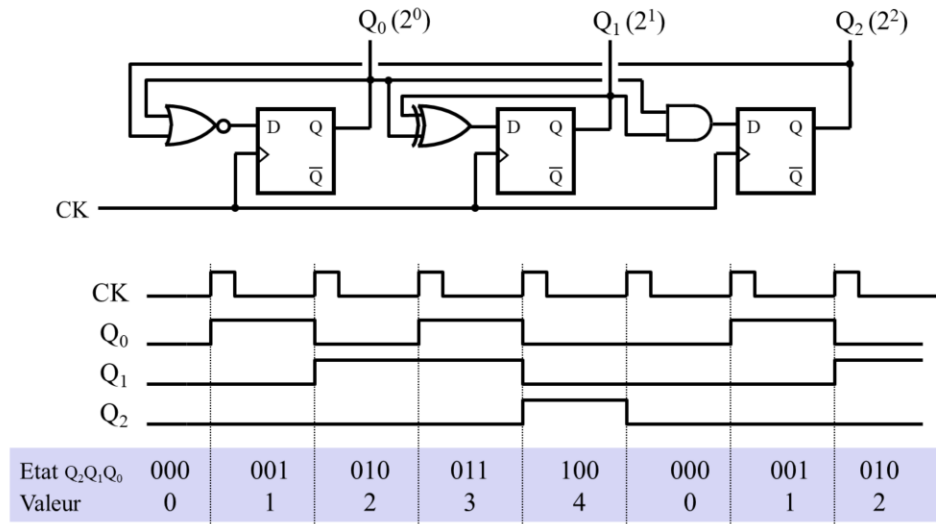
La méthode d'analyse consiste à:

1. Etablir l'équation de chaque D_i , donc de l'état futur Q_i^+ pour chaque bascule en fonction de tous les Q_i .
2. Etablir la table de vérité de l'état futur du système pour tous les états présents possibles (toutes les combinaisons possibles de tous les Q_i).
3. En partant de l'état initial, établir s'enchaînent tous les états possibles du systèmes (le diagramme d'état) en utilisant la table de vérité.

Ou/et construire le chronogramme de la page précédente qui représente l'évolution temporelle des états de chaque bascule.

Dans cet exemple les états 5 (101), 6 (110) et 7 (111) n'apparaissent jamais en fonctionnement normal. Toutefois l'un d'eux peut apparaître lors de l'enclenchement ou suite à un parasite; il est alors important de vérifier que le système retourne dans son cycle normal et ne reste pas piégé dans un cycle parasite.

EXEMPLE 3 : COMPTEUR SYNCHRONES PAR 5



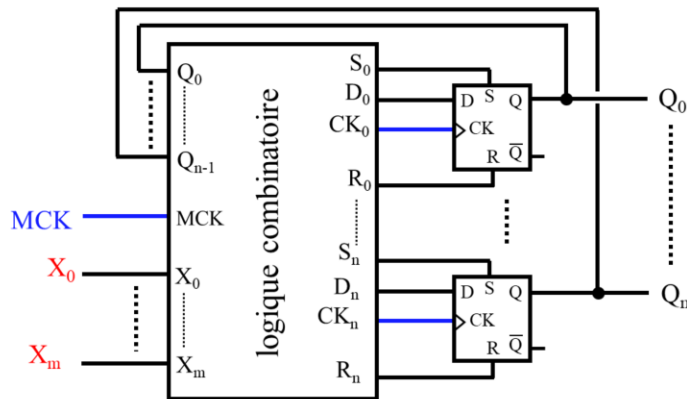
On voit qu'en attribuant un poids 2^i à chaque bit Q_i , l'état du compteur représentera le nombre binaire correspondant :

$$Valeur = \sum_{i=0}^{n-1} b_i 2^i$$

Le circuit compte jusqu'à 4, mais au 'cinquième' CK il va maintenir Q_0 à 0 (au lieu de transiter vers '1') et remettre Q_2 à '0' (tandis qu'il était à '1'). L'état des sorties sera donc 0, 0, 0.

SYSTÈMES LOGIQUES SÉQUENTIELS ASYNCHRONES

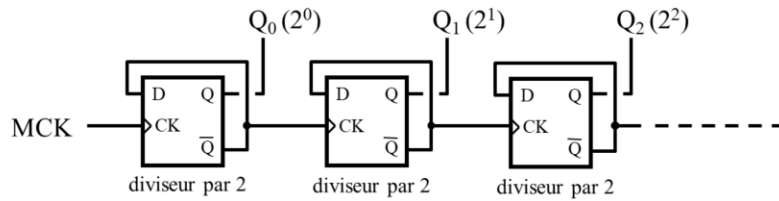
SYSTÈMES LOGIQUES SÉQUENTIELS ASYNCHRONES



Chaque bascule peut avoir son propre signal d'horloge, ou elle peut être forcée à 1 (Set) ou à 0 (Reset) à tout instant autre qu'au front de son horloge. Les bascules ne sont donc pas synchrones.

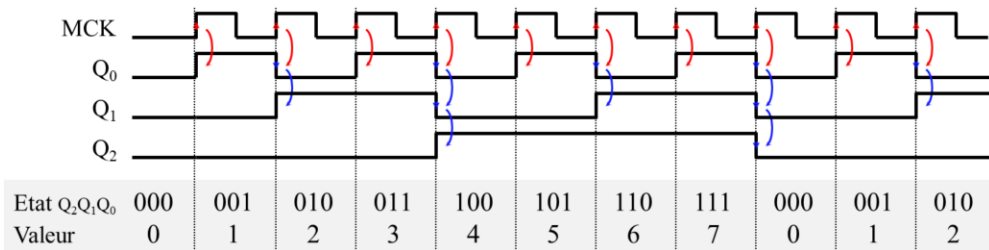
Dans ce système la logique combinatoire génère le signal d'horloge de chaque bascule à partir d'une horloge maître MCK combinées avec des entrées X_j et des sorties Q_i . Souvent la logique combinatoire peut aussi forcer l'état de chaque bascule à tout instant par des commandes supplémentaires Set et Reset. Le système est dit asynchrone car les bascules ne sont plus synchronisées sur le même front d'horloge.

EXEMPLE 1 : COMPTEUR EN CASCADE PAR 2^N



Q_i change au flanc montant de $CK_i = \overline{Q_{i-1}}$, donc au flanc descendant de Q_{i-1}

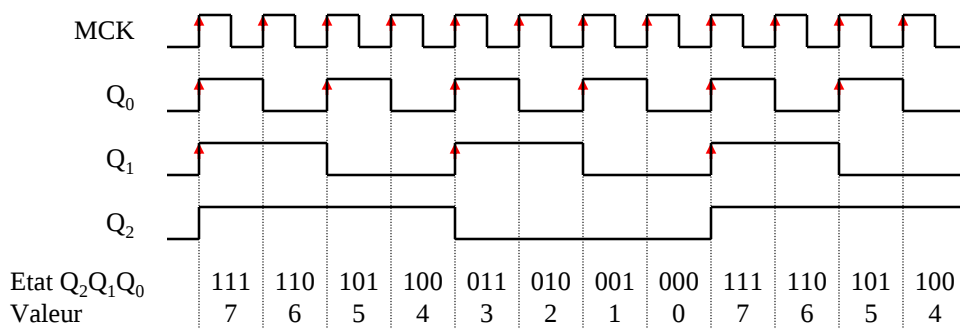
Exemple : $n = 3$, compteur asynchrone par $2^3 = 8$



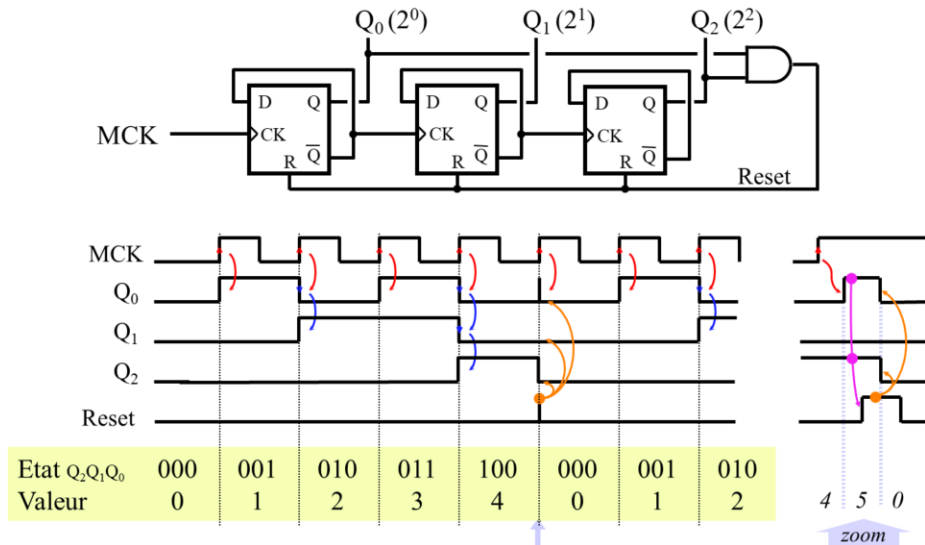
Initiation à l'électronique - Chapitre 7 – Logique et circuits numériques - page 40

Chaque bascule divise par deux la fréquence de son horloge. Comme la sortie d'un diviseur est l'horloge du diviseur suivant, le facteur de division total est donc de $2^{\text{nb de bascules}}$.

On utilise la sortie Q de chaque diviseur comme horloge du suivant, ce qui correspond à déclencher la bascule i au front descendant de Q_{i-1} . Ceci est nécessaire pour que la valeur correspondant à l'état du système aille en croissant. On peut vérifier facilement que si chaque diviseur i est commandé par le front montant de Q_{i-1} on obtient un compte à rebours (down counter).



EXEMPLE 2 : COMPTEUR EN CASCADE PAR 2^N ET RESET A 5



C'est un compteur par 2^3 , remis à 0 dès qu'il arrive à 5 (état transitoire très bref)

Si le compteur est remis quasi instantanément à 0 dès qu'il atteint la valeur m , il va passer par les états : 0, 1, 2, ..., $m-2$, $m-1$, (' m ' n'est pas atteint), 0, 1, 2, ..., ce qui correspond à un compteur par m . Il faut que le compteur original soit de modulo supérieur à m . Le plus simple est de prendre un compteur asynchrone par $2^n \geq m$.

Avec ce principe, on fait facilement un compteur par n'importe quel nombre.

SYNCHRONES VERSUS ASYNCHRONES

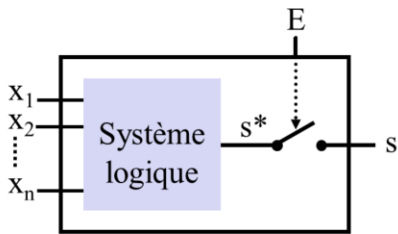
Synchrones

- Temps de propagation du front d'horloge identique pour toutes les bascules: synchronisation des entrées
- Optimum pour la rapidité
- Mais système combinatoire qui peut s'avérer complexe (nombre de combinaisons possibles: $2^{nb \text{ de bascules}}$)

Asynchrones

- Temps de propagation variable d'une entrée, augmente avec le nombre de transitions à réaliser.
- Risques d'états transitoires parasites
- Permet de minimiser la complexité du système combinatoire associé

SORTIE 3 STATES



L'entrée d'activation E (Enable) commande l'interrupteur

s^* est la variable binaire générée par le système logique

s est la sortie à trois états (3-State)

Lorsque l'interrupteur est fermé la sortie est "active" (enabled)

$$s = s^* [0, 1]$$

L'état est imposé par le système logique

Lorsque l'interrupteur est ouvert la sortie est "inactive" (disabled)

La sortie s est dite 'flottante' (High Z)

L'état sera imposé par tout autre sortie active connectée à s

Permet de connecter les sorties à une ligne de bus qui sert aussi de support à d'autres signaux.

Les sorties 3-state (ou three-state) sont souvent représentées par:

