ARCHITECTURE

INFO-SUP

RESUME DE COURS ET CAHIER D'EXERCICES

EPITA **F. GABON**

COURS

EXERCICES

LIVRES D'ARCHITECTURE

- Electronique digitale par P. Cabanis (Dunod)
  Un peu succinct sur les bases de logique mais introduit des notions de programmation, de langages et les microprocesseurs

- Logique combinatoire et technologie par M. Gindre et D. Roux (Ediscience)
  très complet sur les circuits logiques de base y compris l'aspect technologique (TTL, CMOS…), des exos corrigés.

- Logique séquentielle par M. Gindre et D. Roux (Ediscience)
  suite du précédent, mêmes remarques

- Cours et problèmes d'électronique numérique par J.C. Lafont et J.P. Vabre (ellipses)
  bien fait et complet sur le programme de sup

RESUME D'ELECTRONIQUE LOGIQUE

Ces formules sont à connaître par cœur. Les tables de vérité des portes et bascules sont données **en français** et sous une forme pratique et **directement utilisable** : c'est sous cette forme qu'elles sont le plus simples à retenir.

I. PORTES LOGIQUES

ET : **dès** qu'une entrée est à 0 , la sortie est à 0
NON-ET : **dès** qu'une entrée est à 0 , la sortie est à 1
OU : **dès** qu'une entrée est à 1 , la sortie est à 1
NON-OU : **dès** qu'une entrée est à 1 , la sortie est à 0
OU exclusif : **si** les deux entrées sont **différentes**, la sortie est à 1 : $A \oplus B = A.\overline{B} + \overline{A}.B$
NON-OU exclusif : **si** les deux entrées sont **identiques**, la sortie est à 1 (fonction identité) : $\overline{A \oplus B} = A.B + \overline{A}.\overline{B}$

ATTENTION : ne pas confondre le OU logique qui se note par un "+" et l'addition de **nombres** qui se note aussi par un "+".

II. FORMULES ESSENTIELLES

$A + A = A$	$A \,.\, A = A$	$A + A.B = A$	$A + \overline{A}.B = A + B$
$A + \overline{A} = 1$	$A.\overline{A} = 0$	$A.B + \overline{A}.C + B.C = A.B + \overline{A}.C$	
$A + 1 = 1$	$A \,.\, 1 = A$	$\overline{A.B} = \overline{A} + \overline{B}$ (théorème de Morgan)	
$A + 0 = A$	$A \,.\, 0 = 0$	$\overline{A + B} = \overline{A}.\overline{B}$ (théorème de Morgan)	

III. MULTIPLEXEUR 2^n vers 1

n entrées d'adresses affectées d'un poids (c.a.d. formant un nombre binaire) : N …. CBA
2^n entrées de données : $E_0 \ldots E_{2^n-1}$
1 sortie : S telle que si $(N \ldots CBA)_2 = i_{10}$ alors $S = E_i$

IV. DECODEUR n vers 2^n

n entrées d'adresses affectées d'un poids (c.a.d. formant un nombre binaire) : N …. CBA
2^n sorties : $Y_0 \ldots Y_{2^n-1}$ telles que si $(N \ldots CBA)_2 = i_{10}$ alors seule la sortie Y_i est activée

V. BASCULES

Différents types de bascules

1) bascules RS : R = R(eset) ou Cl(ear) ou Mise à 0 ; S = S(et) ou Pr(eset) ou Mise à 1

- état actif sur **l'une** des entrées : la sortie **Q** se met dans l'état demandé.
- aucun état actif : aucun changement : état mémoire
- état actif sur **les deux** entrées : état interdit

2) bascules D

L'état présent sur l'entrée D au moment du front (ou pendant l'état **actif** de l'entrée d'horloge) est recopié sur la sortie Q (sauf si l'une des entrées de forçage à 0 ou à 1 est active)

3) bascules JK

Les changements d'état des sorties se font (éventuellement)au moment du front (sauf si l'une des entrées de forçage est active : voir ci-dessous)

J = K = 0	: Q ne change pas d'état : état mémoire
J ≠ K	: Q prend l'état de J , Q celui de K
J = K = 1	: Q change forcément d'état (TOGGLE dans les docs)

VI. SYNCHRONISATION

Il existe 3 façons différentes de synchroniser les changements d'état des sorties par rapport à l'entrée d'horloge.

1) Synchronisation sur niveau
Tant que l'horloge est dans l'état actif ("1" en général), les sorties "réagissent" immédiatement aux changements d'état des entrées.

2) Synchronisation sur front
Les sorties ne changent d'état **qu'au moment** du front actif sur l'entrée d'horloge.
Ce front peut être montant (passage de "0" à "1") ou descendant (passage de "1" à "0").
Sur le schéma des bascules, ce type de synchronisation est représenté par un petit triangle sur l'entrée d'horloge, associé de plus à un petit rond si le front actif est descendant.

3) Synchronisation sur impulsion (bascules dites "maître-esclave")
Sur le front montant, la bascule **mémorise** l'état des entrées : les sorties ne changent pas.
Sur le front descendant, les sorties changent (éventuellement) d'état en fonction de l'état des entrées **présent au moment du front montant**.

L'intérêt de ce dernier type de synchronisation est de dissocier l'analyse de l'action : si la durée du créneau d'horloge est supérieure au temps de retard des bascules (donné par le constructeur), aucun mauvais fonctionnement ne peut être généré par ces temps de retard (comme cela était possible avec les bascules synchronisées sur front).

VII. ENTREES DE FORCAGE ASYNCHRONES

L'état actif (en général 0) sur l'une des 2 entrées de forçage fait **immédiatement** passer la sortie Q dans l'état demandé, sans tenir compte de l'horloge.

L'état actif sur l'entrée R(eset) (aussi appelée Clear) force Q à 0.
L'état actif sur l'entrée Pr(eset) force Q à 1.
L'état actif sur les 2 entrées de forçage est évidemment interdit !

SYSTEMES DE NUMERATION

I. Système décimal et définitions

Quand on voit le nombre 537, on sait que le chiffre 5 correspond aux centaines, le chiffre 3 aux dizaines et le chiffre 7 aux unités.

On peut écrire ce nombre sous la forme d'un polynôme : $537 = 5.10^2 + 3.10^1 + 7.10^0$.

- La **base** dans lequel ce nombre est écrit est la base 10 car nous avons … 10 doigts (si, si …)
  En base 10, il existe 10 symboles (appelés chiffres) de 0 à 9.

- Le **rang** d'un chiffre est par définition sa position dans le nombre en partant du rang 0 et en commençant par le chiffre de droite (celui des unités)
  Dans l'exemple ci-dessus, le rang de 7 est 0, celui de 3 est 1 et celui de 5 est 2.

  Mais, **vu sa place**, le chiffre 5 "pèse" plus lourd que le chiffre 7 bien que sa valeur propre soit plus petite.

- Le **poids** d'un chiffre x est la base élevée à la puissance de son rang : poids (x) = $10^{\text{rang(x)}}$.
  Le chiffre de droite s'appelle le chiffre de poids faible (pf) et celui de gauche le chiffre de poids fort (PF).

  7 est le chiffre de poids faible : son poids est 1 (10^0)
  5 est le chiffre de poids fort : son poids est 100 (10^2)

Généralisation à un nombre de n + 1 chiffres : $\boxed{N = \sum_{i=0}^{n} a_i.10^i}$ avec $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

II. Base b quelconque

Si l'on n'avait que 8 doigts, on ne compterait sûrement pas en base 10 mais en base 8, on parlerait de "huitaines" de "soixante-quatraines" etc… , il n'y aurait que 8 symboles (de 0 à 7) et le poids d'un chiffre x vaudrait $8^{\text{rang(x)}}$ mais, à cela près, toutes les définitions resteraient identiques.

On peut donc écrire de façon générale : $\boxed{N = \sum_{i=0}^{n} a_i.b^i}$ avec $a_i \in \{0, 1, 2, \ldots (b-1)\}$

Un symbole devant correspondre à **un chiffre**, si la base est supérieure à 10, on prendra par convention comme symboles supplémentaires les lettres majuscules de l'alphabet en commençant par A.

Ainsi, en base 2 on aura : $a_i \in \{0, 1\}$
En base 8 on aura : $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
En base 12 on aura : $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B\}$
En base 16 on aura : $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

En dehors de la base 10 (le système décimal), les 3 bases les plus utilisées en informatique sont :

- La base 2 : le système est dit binaire (chaque chiffre binaire 0 ou 1 est appelé BIT de l'anglais BInary digiT)
- La base 8 : le système est dit octal
- La base 16 : le système est dit hexadécimal

La base se note en indice après le nombre. Toutefois on peut ne pas la noter en bases 2, 10 ou 16, **à condition qu'il n'y ait aucun risque d'erreur**.

ex : 100 est ambigu (100_{10} ou 100_2 soit 4_{10}) sauf si le contexte est évident.

Il faut donc retenir

$A_{16} = 10_{10}$; $B_{16} = 11_{10}$; $C_{16} = 12_{10}$; $D_{16} = 13_{10}$; $E_{16} = 14_{10}$; $F_{16} = 15_{10}$

ex : $537_8 = 5.8^2 + 3.8^1 + 7.8^0$; $537_{12} = 5.12^2 + 3.12^1 + 7.12^0$; $1101_2 = 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0$

$A3F91C_{16} = A.16^5 + 3.16^4 + F.16^3 + 9.16^2 + 1.16^1 + C.16^0$

mais 5367_6 est incorrect à cause du 6 et du 7 qui ne peuvent pas exister en base 6.

III. Changements de base (nombres entiers)

1. Conversion d'un nombre en base quelconque vers la base 10

Il suffit d'utiliser la forme polynomiale vue ci-dessus, de remplacer éventuellement les lettres par leurs valeurs décimales et d'effectuer l'addition.

ex : $537_8 = 5.8^2 + 3.8 + 7 = 5.64 + 3.8 + 7 = 351_{10}$ $\quad$ $ABF_{16} = 10.16^2 + 11.16 + 15 = 2751_{10}$

2. Conversion d'un nombre décimal en base quelconque

- Diviser le nombre décimal par la base dans laquelle on veut le convertir.
- Répéter l'opération jusqu'à ce que le quotient soit nul.
- Ecrire tous les restes en prenant pour poids fort le dernier reste obtenu.
- Si la base d'arrivée est > 10, convertir les restes > 9 en lettres.

ex : on veut convertir 351_{10} en base 8

351 / 8 = 43	reste 7		
43 / 8 = 5	reste 3	⇨	$351_{10} = 537_8$ … comme on s'y attendait !
5 / 8 = 0	reste 5		

Cette règle, appliquée à la conversion en binaire des nombres décimaux de 0 à 15 donne les résultats suivants qu'il faut connaître par cœur car ils seront très utilisés.

base 10	base 16	base 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Remarques : avec un peu d'habitude, il est facile de faire les conversions de tête, en se souvenant des 4 premiers poids en binaire à savoir : 8 4 2 1

ex : $13_{10} = 8 + 4 + 1$ mais il manque le 2

Donc on met des 1 à la place des chiffres qui existent dans la décomposition et des 0 à la place de ceux qui n'existent pas.

Cela donne : $13_{10} = 1101_2$

Dans l'autre sens ... c'est l'inverse :

ex : 1001_2 : le 1 de gauche "pèse" 8 et celui de droite "pèse" 1 : donc $1001_2 = 8 + 1 = 9_{10}$

Par contre pour les lettres en hexa, pas d'astuce, il faut les apprendre !

3. Conversion rapide entre le binaire, l'octal et l'hexadécimal

Ces bases ont la particularité d'être des puissances de 2

en octal, chaque chiffre est écrit sur 3 bits de 000 à 111
en hexadécimal, chaque chiffre est écrit sur 4 bits de 0000 à 1111

- Donc pour convertir un nombre octal ou hexadécimal en binaire, il suffit de convertir chacun de ses chiffres en utilisant le tableau ou la méthode ci-dessus.

Attention : chaque chiffre octal devra être écrit avec 3 bits, et chaque chiffre hexadécimal devra l'être avec 4.

ex : $3506_8 = 011101000110_2$ mais $3506_{16} = 0011010100000110_2$

Rem : les 0 à gauche peuvent être omis mais le 0 de rang 1 dans 3506 se code sur 3 "0" binaires si le nombre était en octal et sur 4 "0" si le nombre était en hexadécimal.

- Pour convertir un nombre binaire en octal ou hexadécimal, on découpe les bits par paquets de 3 (pour l'octal) ou de 4(pour l'hexadécimal) … en partant de la droite!

ex : $11|100|101|110|011|000|011_2 = 3456303_8$
mais le même nombre binaire donne en base 16 : $1110|0101|1100|1100|0011_2 = E5CC3_{16}$

Ces transformations rapides sont pratiques et très utilisées en particulier quand on programme en assembleur.

4. Conversion d'une base quelconque vers une autre base quelconque

La seule solution est de passer par le décimal sauf si les 2 bases sont des puissances de 2 ; dans ce cas, il est plus simple de passer par le binaire comme il a été vu ci-dessus.

IV. Changements de base (nombres fractionnaires)

1. Conversion d'un nombre en base quelconque vers la base 10

On peut étendre la forme polynomiale aux puissances négatives :

ex : $537,28_{10} = 5.10^2 + 3.10^1 + 7.10^0 + 2.10^{-1} + 8.10^{-2}$
$537,28_{16} = 5.16^2 + 3.16^1 + 7.16^0 + 2.16^{-1} + 8.16^{-2}$

La méthode de conversion sera la même

$537,2C_{16} = 5.16^2 + 3.16^1 + 7.16^0 + 2.16^{-1} + 12.16^{-2} = 1335,171875_{10}$

2. Conversion d'un nombre décimal en base quelconque

Pour la partie entière, on procède comme pour les nombres entiers par divisions successives.

Pour la partie fractionnaire, on remplace les divisions par des multiplications :

- Multiplier la partie fractionnaire du nombre décimal par la base dans laquelle on veut la convertir.
- Répéter l'opération pour atteindre la précision voulue.
- Ecrire toutes les parties entières dans l'ordre dans lequel on les a obtenues.
- Si la base d'arrivée est > 10, convertir les parties entières > 9 en lettres.

ex : on veut convertir $351,943_{10}$ en base 8

0,943 * 8 = 7,544	on garde 7	⇨	$0,943_{10} = 0,7426_8$
0,544 * 8 = 4,352	on garde 4		
0,352 * 8 = 2,816	on garde 2		
0,816 * 8 = 6,528	on garde 6		

En définitive $351,943_{10} = 537,7426_8$

Vérification : si l'on retransforme $0,7426_8$ en décimal on obtient 0,9429 soit pratiquement 0,943
Si l'on s'était contenté de 3 chiffres après la virgule (soit 0,742) on aurait obtenu 0,9414

On voit que plus la base est petite, plus il faut de décimales pour conserver la précision initiale.

En base 10, 3 chiffres après la virgule correspondent à une précision de $1/10^3$ soit 0,001.
En base 8, 3 chiffres après la virgule correspondent à une précision de $1/8^3$ soit ≈ 0,002.

Si l'on veut convertir un nombre décimal en binaire avec une précision du millième, il faut aller jusqu'à 10 chiffres binaires après la virgule (en effet $1/2^{10} = 1/1024 \approx 0,001$).

V. OPERATIONS ARITHMETIQUES

De façon générale, les règles habituelles utilisées en décimal ne changent pas mais bien sur il faut tenir compte de la base de travail :

Ainsi, en base 16 : 9 + 4 = D et 9 + C = 15 (ou 5 et une retenue de 1 vers la colonne de gauche)

Jusqu'ici, nous n'avons traité que les nombres positifs et on pourrait imaginer de traiter les nombres négatifs en mettant un signe moins devant comme d'habitude mais l'informatique, qui ne connaît que les 1 et les 0, traite différemment les nombres négatifs.

1. Complément à 1

En décimal, on forme le complément à 9 d'un nombre en remplaçant chaque chiffre de ce nombre par sa différence avec 9.

ex : le complément à 9 de 6473, noté $C_9(6473)$ est 3526

En binaire, on forme le complément à 1 d'un nombre en remplaçant chaque chiffre de ce nombre par sa différence avec 1 (on remplace les 1 par des 0 et réciproquement).

ex : le complément à 1 de 11010, noté $C_1(11010)$ est 00101

2. Complément à 2 (ou complément vrai)

En décimal, on forme le complément à 10 d'un nombre en remplaçant le chiffre des unités par sa différence avec 10 et les autres chiffres de ce nombre par leur différence avec 9.

ex : le complément à 10 de 6473, noté $C_{10}(6473)$ est 3527

On peut aussi dire que le complément à 10 d'un nombre s'obtient en le soustrayant de la puissance de 10 immédiatement supérieure à ce nombre : en effet 3527 = 10000 - 6473

On voit que le complément à 10 s'obtient en ajoutant 1 au complément à 9

En binaire, on va utiliser la règle ci-dessus et former le complément à 2 d'un nombre en ajoutant 1 au complément à 1

ex : le complément à 2 de 11010, noté $C_2(11010)$ est 00101 + 1 = 00110

On peut aussi dire que le complément à 2 d'un nombre s'obtient en le soustrayant de la puissance de 2 immédiatement supérieure à ce nombre : en effet 00110 = 100000 - 11010

Une astuce permet d'éviter cette addition : pour obtenir le complément à 2 d'un nombre, on conserve tous les bits à partir de la droite jusqu'au 1^{er} 1 **compris** et on inverse tous les autres.

3. Soustraction par complémentation à 2

Le but est de remplacer la soustraction par une addition

a) En décimal

Soit à faire la soustraction suivante 742 – 568 (= 174)

Comme 568 = 1000 – 432, on peut écrire 742 – 568 = 742 – (1000 – 432) = 742 + 432 – 1000 = 174

Comme 432 est le complément à 10 de 568, on a en fait effectué l'opération suivante :

$x - y = x + C_{10}(y) -$ (puissance de 10 immédiatement supérieure à x)

Au lieu de soustraire 1000, il aurait suffit de négliger la dernière retenue dans le résultat.

En effet 742 + 432 = 1174 = 174 si on néglige le 1 de gauche

En résumé, au lieu de soustraire un nombre y d'un nombre x, on ajoute à x le complément à 10 de y et on oublie le (n + 1)ième chiffre si on travaille sur n chiffres.

Rem : si les 2 nombres x et y ne comportent pas le même nombre de chiffres, il faut rajouter des 0 devant y avant de chercher son complément à 10.

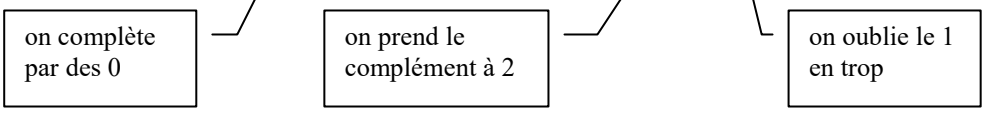
ex : 742 – 67 = 742 – 067 = 742 + 933 = (1)675 = 675

b) En binaire

C'est pareil !! (avec la même remarque que ci-dessus)

$x - y = x + C_2(y) -$ (puissance de 2 immédiatement supérieure à x)

11100101 – 110111 = 11100101 – 00110111 = 11100101 + 11001001 = (1)10101110 = 10101110



Mais, pour l'instant, la soustraction x – y n'est toujours pas faisable si x < y
(sauf en effectuant y – x et en mettant moins devant le résultat !)

4. Normalisation et nombres signés

Comme l'ordinateur ne comprend pas les signes + et -, on va les représenter par un bit de la façon suivante :

On commence par normaliser la taille à un nombre précis de bits par exemple 8 soit un octet.
On décide que le **bit de gauche** sera égal à 0 si le nombre est positif ou nul et à 1 si le nombre est négatif. Ce bit est appelé **bit de signe** et les nombres codés ainsi s'appellent **nombres signés**.

On applique les règles suivantes :

a) Conversion décimal vers binaire signé

- Si x_{10} est positif ou nul, on le convertit normalement en binaire.
- Si x_{10} est négatif, on convertit sa valeur absolue en binaire et on en prend le complément à 2.

ex : 68 = 0100 0100 ; - 67 : |- 67| = 0100 0011 donc – 67 = C_2(0100 0011) = 1011 1101

b) Conversion binaire signé vers décimal

- Si le bit de signe de $x_2 = 0$, on le convertit directement en décimal
- Si le bit de signe de $x_2 = 1$, cela veut dire que x_{10} est négatif : on cherche le complément à 2 de x_2, on le convertit en décimal et on rajoute un moins devant !

ex : 0111 0100 = 116 ; 1010 0010 : le bit de signe = 1 : le nombre est négatif : on en prend le complément à 2 soit 0101 1110. On le convertit en décimal soit 94. Donc 1010 0010 = - 94

c) Limites

Comme on s'est limité à 8 bits, on ne pourra coder en binaire que 256 valeurs différentes.

En nombres non signés (donc exclusivement positifs) on pouvait aller de 0_{10} (= 0000 0000_2) à 255_{10} (= 1111 1111_2) soit 256 valeurs en tout car $256 = 2^8$.

En nombres signés, le plus grand nombre commençant par un 0 (donc positif) est 0111 1111 = 127

Le nombre binaire suivant est 1000 0000 mais, comme il commence par un 1, il doit être considéré comme négatif : pour trouver sa valeur décimale, on en prend le complément à 2 soit 1000 0000 (c'est le même !) qui, converti en décimal, donne 128.
128 est donc la valeur absolue de 1000 0000 et on obtient en définitive 1000 0000 = - 128

Sur un octet les 256 valeurs codables en nombres signés vont donc de – 128 à + 127.

Le nombre 128 n'est pas codable en nombres signés sur 8 bits.

En nombres signés sur n bits, on peut coder les nombres de -2^{n-1} à $2^{n-1} - 1$.

Le tableau suivant regroupe quelques valeurs particulières des nombres signés et fait aussi apparaître les nombres hexadécimaux correspondants.

base 10	base 2	base 16
127	0111 1111	7F
126	0111 1110	7E
2	0000 0010	02
1	0000 0001	01
0	0000 0000	00
-1	1111 1111	FF
-2	1111 1110	FE
-127	1000 0001	81
-128	1000 0000	80

On constate que les valeurs positives sont codées en hexadécimal de 00 à 7F et que les valeurs $\geq 80_{16}$ correspondent à des nombres négatifs.

d) Opérations en nombres signés

Puisqu'on sait maintenant noter les nombres négatifs, on peut effectuer la soustraction x – y avec x < y : le résultat est négatif, son bit de signe est à 1 et il suffit de le convertir comme vu ci-dessus.

ex : 0011 11 00 – 0101 0101 = 0011 1100 + 1010 1011 = 1110 0111 qui correspond à un nombre négatif à cause du 1 de gauche.
Pour calculer sa valeur décimale, on en prend le complément à 2 soit 0001 1001 et on le convertit en décimal soit 25. Le résultat de la soustraction est donc – 25.

Attention : dans la mesure où les nombres sont codés sur 8 bits, le résultat doit rester compris dans l'intervalle – 128 … + 127. Un dépassement doit générer une erreur.

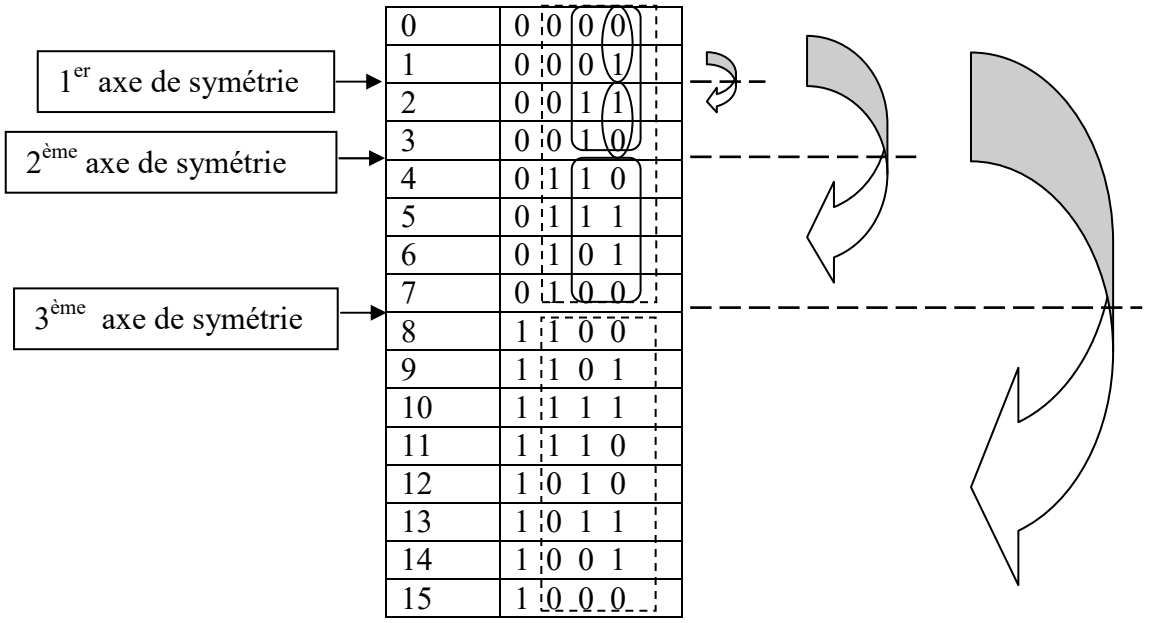
VI. Différents types de codes

Jusque là, nous avons travaillé avec le code binaire dit naturel qui est le plus simple et le plus pratique pour effectuer des opérations arithmétiques, mais, en informatique, il existe différentes façons de représenter les nombres selon les besoins et les contraintes. Nous allons en voir quelques unes.

1. Code Gray

En code binaire naturel, quand on passe de 7 à 8, donc de 0111 à 1000, les quatre bits changent à la fois. Cela peut être gênant car les changements ne sont jamais totalement simultanés et de très faibles écarts peuvent entraîner des erreurs. Le code Gray élimine ce risque car, dans ce code, un seul bit change à la fois entre deux valeurs consécutives. On parle alors de code à **distance unité**.

On l'appelle aussi **code réfléchi** car sa construction s'opère par réflexion au sens géométrique du terme, c'est à dire par "pliage" autour d'un axe de réflexion.



2. Code BCD (Binary Coded Decimal)

Pour l'affichage des nombres, le code binaire naturel n'est pas pratique : en effet 13 se code en binaire naturel 1101 mais pour l'afficher par exemple sur une calculatrice, il faut séparer le 1 du 3 et envoyer chaque chiffre sur l'afficheur à cristaux liquides correspondant.

D'où l'idée qui consiste à coder chaque chiffre décimal sur 4 bits : c'est le code BCD. C'est le même que le code binaire naturel mais il s'arrête à 9.
Pour coder 10, on code le 1 puis le 0 sur 4 bits : 10_{10} = 0001 0000 en BCD

ex : 3509 = 0011 0101 0000 1001

Rem : les "0" apparaissent même en poids fort car chaque chiffre décimal doit être codé sur 4 bits.

Par contre ce code n'est pas adapté aux calculs numériques.

3. Code ASCII (American Standard Code for Information Interchange)

C'est un standard universel pour normaliser les échanges entre machines.

ex : a = 0110 0001 ; A = 0100 0001 ; 0 = 0011 0000 ; ESC(ape) = 0001 1011 etc…

4. Autres codes

Il existe d'autres codes qui ont chacun leur utilité.
Par exemple, le code EXCESS 3 (qui consiste à ajouter 3 au code BCD) a été utilisé pour faire des calculs numériques directement en BCD.

Les codes détecteurs d'erreurs ou détecteurs et correcteurs d'erreurs sont très utilisés dans les transmissions.

ALGEBRE DE BOOLE

C'est une algèbre un peu particulière et apparemment très simple dans la mesure où les **variables** (dites **booléennes**) qu'elle utilise ne peuvent prendre que 2 états, qu'on note de façon conventionnelle 0 et 1. Et pourtant, c'est avec cela qu'on fait toute l'informatique !!

Cette algèbre est, de plus, munie de 3 **opérateurs** NON, ET et OU que nous allons définir.

De même qu'en mathématiques, l'application d'opérateurs à ces variables donne des **fonctions** qui sont elles aussi booléennes dans la mesure où elles ne peuvent prendre que les valeurs 0 ou 1.

ex : X et Y étant des variables booléennes, Z = X ET Y est une fonction booléenne de X et de Y.

I. Opérateurs de base

1. Généralités

Les opérateurs sont définis de façon conventionnelle par une "table de vérité", c'est à dire un tableau qui donne la valeur de la fonction en fonction de l'état de la (ou des) variable(s) d'entrée.

S'il n'y a qu'une seule variable d'entrée, deux cas sont possibles (0, 1) et la table de vérité comprendra 2 lignes.
S'il y a 2 variables d'entrée, quatre cas sont possibles (00, 01, 10, 11) et la table de vérité comprendra 4 lignes.
De façon générale, s'il y a n variables d'entrées, la table de vérité comprendra 2^n lignes.
Dans les tables de vérité, il est d'usage de représenter les différents cas possibles les uns en dessous des autres en utilisant l'ordre binaire naturel vu précédemment.

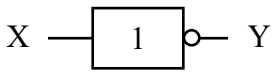
Les opérateurs sont représentés par un schéma en forme de rectangle qui contient un symbole significatif de l'opérateur considéré et qui se lit de gauche à droite : la ou les entrées sont notées à gauche du rectangle et la sortie à droite.

On les appelle souvent "portes logiques" car ils peuvent laisser passer une information ("porte ouverte") ou l'empêcher de passer ("porte fermée").

2. Opérateur NON : Y = NON X ou $Y = \overline{X}$

X	Y
0	1
1	0

Table de vérité



Symbole

On voit dans la table de vérité que si X = 1, Y = 0 et réciproquement. Cet opérateur est parfois appelé inverseur, il effectue le complément à 1 d'une variable d'entrée.
On dit Y = NON X ou X barre car on note la complémentation par une barre au dessus de la variable que l'on veut complémenter : $Y = \overline{X}$
Nous allons profiter de cet opérateur (le plus simple qui existe) pour définir sur un plan électrique ce qu'on appelle des 0 et des 1.

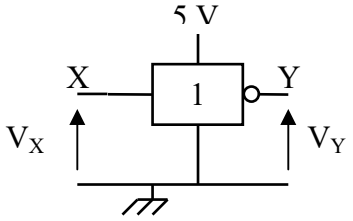
Les composants électroniques qui forment les ordinateurs sont appelés **Circuits Intégrés** (en abrégé C.I.) car ils … "intègrent" (= regroupent) un certain nombre de circuits élémentaires.

Pour pouvoir fonctionner, ces composants sont alimentés en énergie par une … "alimentation" qui transforme la tension alternative du secteur (220 V à 50 Hz) en une tension continue.
Une technologie un peu ancienne mais encore utilisée s'appelle la technologie TTL pour Transistor Transistor Logic et nous allons la prendre pour exemple.
Dans celle-ci, la tension d'alimentation est fixée à 5 V, c'est à dire que les C.I. sont alimentés par une tension de 5 V sur leur borne + par rapport à leur borne – qui est reliée à la masse.

En logique dite "positive", un niveau 1 logique correspond à une tension de 5 V par rapport à la masse et un niveau 0 logique correspond à une tension de 0 V par rapport à la masse.
(La logique dite "négative" correspond à l'inverse mais elle est moins utilisée et nous la laisserons de coté)

Cela donne un schéma complet faisant apparaître l'alimentation et les niveaux de tension :

$X = 1 \Leftrightarrow V_X = 5\ V$
$X = 0 \Leftrightarrow V_X = 0\ V$
$Y = 1 \Leftrightarrow V_Y = 5\ V$
$Y = 0 \Leftrightarrow V_Y = 0\ V$



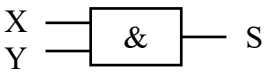
Rem : on peut aussi faire une analogie avec des circuits électriques formés uniquement d'interrupteurs dans lesquels le courant passe ou non selon qu'ils sont fermés ou ouverts, mais, en architecture des systèmes informatiques, on utilise plutôt les niveaux de tension.

Dans les schémas usuels, l'alimentation et les tensions ne sont pas représentées. Seuls apparaissent les noms des variables logiques et leurs valeurs logiques (0 ou 1) mais n'oubliez pas que "derrière" tout cela il y a bien des Volts et des Ampères !

3. Opérateur ET (ou AND) : S = X ET Y = X.Y = XY

X	Y	S
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité



Symbole

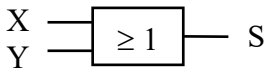
Dès, qu'une entrée est nulle, la sortie est nulle, ou si l'on veut le dire autrement, il faut que les 2 entrées soient à 1 pour que la sortie soit à 1.

C'est l'équivalent du produit en mathématiques : dès qu'un facteur est nul, le produit est nul.
C'est pour cela qu'on peut remplacer le mot ET par un point ou même l'omettre complètement.

4. Opérateur OU (ou OR) : S = X OU Y = X + Y

X	Y	S
0	0	0
0	1	1
1	0	1
1	1	1

Table de vérité



Symbole

Le symbole ≥ 1dans le rectangle rappelle que S = 1 dès que le nombre d'entrées à 1 est ≥ 1.

Dès, qu'une entrée est à 1, la sortie est à 1, ou si l'on veut le dire autrement, il faut que les 2 entrées soient nulles pour que la sortie soit nulle.

Ce n'est pas l'équivalent de la somme en mathématiques : en effet 1 OU 1 = 1 alors qu'en math 1 + 1 = … 2 ! mais par analogie avec la fonction ET appelée produit, on convient de remplacer la notation OU par le symbole + et on parle quand même de somme.

Rem : dans certains livres, on trouve ∪ (union) à la place de OU et ∩ (intersection) à la place de ET, ce qui rappelle la notation en termes d'ensembles et lève l'ambiguité ci-dessus, mais nous adopterons les notations "+" et ".", plus usuelles en architecture.

5. Propriétés

Les opérateurs vus ci-dessus respectent les lois suivantes, qu'il est facile de vérifier par **induction parfaite**, c'est à dire en vérifiant tous les cas possibles à partir des tables de vérité.

Rem : comme en math, le produit a priorité sur la somme, ce qui permet d'éliminer des parenthèses.

a) Fermeture

(X.Y) et (Y + Y) sont des variables booléennes.

b) Commutativité

X.Y = Y.X X + Y = Y + X

c) Associativité

X.(Y.Z) = (X.Y).Z = X.Y.Z X + (Y + Z) = (X + Y) + Z = X + Y + Z

d) Distributivité de ET sur OU et de OU sur ET

X.(Y + Z) = X.Y +X.Z X + Y.Z = (X + Y).(X + Z)

e) Idempotence (variables identiques)

X.X = X X + X = X

f) Complémentarité

$X.\overline{X} = 0$ $X + \overline{X} = 1$

g) Identités remarquables

$1.X = X$ $0.X = 0$ $1 + X = 1$ $0 + X = X$

Toutes ces formules permettent déjà de simplifier des fonctions logiques, c'est à dire d'éliminer au maximum les opérateurs logiques (sans bien sur modifier la fonction initiale !)

ex : Soit S à simplifier	$S = (X + \overline{Y}).(X + Y) + Z.(\overline{X} + Y)$
par distributivité	$S = (X + \overline{Y}).X + (X + \overline{Y}).Y + Z.(\overline{X} + Y)$
par distributivité	$S = X.X + \overline{Y}.X + X.Y + \overline{Y}.Y + Z.\overline{X} + Z.Y$
par idempotence $(X.X = X)$	$S = X + \overline{Y}.X + X.Y + \overline{Y}.Y + Z.\overline{X} + Z.Y$
par complémentarité $Y.\overline{Y} = 0$	$S = X + \overline{Y}.X + X.Y + Z.\overline{X} + Z.Y$
par identité remarquable $(1.X = X)$	$S = 1.X + \overline{Y}.X + X.Y + Z.\overline{X} + Z.Y$
par distributivité	$S = X.(1 + \overline{Y} + Y) + Z.\overline{X} + Z.Y$
par identité remarquable (sur + puis .)	$S = X + Z.\overline{X} + Z.Y$

Nous pourrons même simplifier encore la fonction S quand nous aurons vu quelques autres formules un peu plus "puissantes" que celles vues ci-dessus.

II. Opérateurs complémentaires

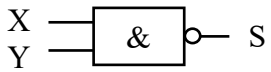
Ces opérateurs peuvent être formés à partir des opérateurs de base mais ils existent déjà en tant que circuits intégrés et permettent le plus souvent de simplifier des systèmes informatiques.

En particulier, les 2 premiers cités (le NAND et le NOR) sont appelés **portes universelles** car ils permettent de recréer n'importe quelle autre porte en les associant entre eux.

1. Opérateur NON-ET (ou NAND) : $S = \overline{X.Y}$

X	Y	S
0	0	1
0	1	1
1	0	1
1	1	0

Table de vérité



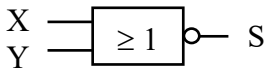
Symbole

Dès, qu'une entrée est nulle, la sortie est à 1, ou si l'on veut le dire autrement, il faut que les 2 entrées soient à 1 pour que la sortie soit nulle.

2. Opérateur NON-OU (ou NOR) : $S = \overline{X + Y}$

X	Y	S
0	0	1
0	1	0
1	0	0
1	1	0

Table de vérité



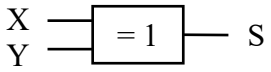
Symbole

Dès, qu'une entrée est à 1, la sortie est nulle, ou si l'on veut le dire autrement, il faut que les 2 entrées soient nulles pour que la sortie soit à 1.

3. Opérateur OU exclusif (ou XOR) : $S = X \oplus Y$

X	Y	S
0	0	0
0	1	1
1	0	1
1	1	0

Table de vérité



Symbole

La sortie est à 1 si l'une des 2 entrées (mais pas les 2) est à 1 ou si l'on veut le dire autrement, la sortie est à 1 si les 2 entrées ne sont pas dans le même état.

Rem : avec des NON, des OU et des ET le OU exclusif s'écrit $S = X.\overline{Y} + \overline{X}.Y$
et il faut le repérer dans une équation pour pouvoir remplacer cette formule un peu compliquée (2 NON, 2 ET et 1 OU) par un simple OU exclusif.

III. Quelques formules complémentaires mais très utilisées

1. Théorème de De Morgan

$\boxed{\overline{X + Y} = \overline{X}.\overline{Y}}$: l'inverse de la somme est égal au produit des inverses

$\boxed{\overline{X.Y} = \overline{X} + \overline{Y}}$: l'inverse du produit est égal à la somme des inverses

L'intérêt de ces formules est aussi technique : il est plus simple dans un système informatique d'utiliser une seule porte NAND (ou NOR) plutôt que 2 inverseurs plus une porte OU (ou ET) pour réaliser la même fonction.

2. Autres formules pratiques

- Formule N° 1 : $\boxed{X + X.Y = X}$: dans une somme, on peut éliminer tous les multiples d'un terme fondamental.

  Démonstration : $X + X.Y = X.1 + X.Y = X.(1 + Y) = X.1 = X$

- Formule N° 2 : $\boxed{X + \overline{X}.Y = X + Y}$: dans la somme d'un terme et d'un multiple de son complément, on peut éliminer le complément.

  Démonstration : on utilise la 1ère formule en remplaçant X par X + X.Y, soit :

  $$X + \overline{X}.Y = X + X.Y + \overline{X}.Y = X + Y.(X + \overline{X}) = X + Y.1 = X + Y$$

  Rem : cette formule est à retenir car elle n'est pas intuitive comme la précédente : en effet il faut d'abord compliquer la formule pour la simplifier ensuite.

- Formule N° 3 : $\boxed{X.Y + \overline{X}.Z + Y.Z = X.Y + \overline{X}.Z}$

  Démonstration : on peut toujours multiplier une variable par 1 sous la forme $X + \overline{X}$ soit :

  $$X.Y + \overline{X}.Z + Y.Z = X.Y + \overline{X}.Z + Y.Z.(X + \overline{X}) = X.Y + \overline{X}.Z + Y.Z.X + Y.Z\overline{X} = X.Y + \overline{X}.Z$$

  car le 3ème terme est un multiple du 1er et le 4ème est un multiple du 2ème (formule N°1).

  Même remarque que ci-dessus

On peut maintenant continuer à simplifier l'équation vue en exemple.

Nous étions arrivés à vérifier : $S = (X + \overline{Y}).(X + Y) + Z.(\overline{X} + Y) = X + Z.\overline{X} + Z.Y$

La formule N° 2 donne : $S = X + Z + Z.Y$
La formule N° 1 donne : $S = X + Z$

En définitive, $(X + \overline{Y}).(X + Y) + Z.(\overline{X} + Y) = X + Z$, ce qui est quand même plus simple !

3. Dualité de l'algèbre de Boole

Des égalités logiques restent vraies si l'on remplace les 1 par des 0, les ET par des OU et réciproquement.

Par exemple : $X.\overline{X} = 0$ donne par dualité : $X + \overline{X} = 1$
$X + \overline{X}.Y = X + Y$ donne par dualité : $X.(\overline{X} + Y) = X.Y$

C'est ce qu'on appelle le principe de dualité de l'algèbre de Boole.

IV. Résolution de problèmes par un système informatique

Le but de l'architecture des systèmes c'est de construire des systèmes à base de composants logiques qui "répondent à la demande", c'est à dire qui effectuent bien le travail (= la ou les fonctions) demandé dans le cahier des charges. Ces systèmes doivent bien sur fonctionner correctement mais, de plus, ils doivent être optimisés, autrement dit, ils doivent utiliser le moins possible de composants logiques car ceux-ci coûtent cher, tiennent de la place sur les circuits imprimés, consomment du courant, tombent parfois en panne …

Pour y parvenir il faut respecter les 3 étapes suivantes :

- Traduire le problème donné dans un table de vérité de même qu'en math on traduit par une équation ou un système d'équations un problème énoncé en français.

- Déduire de cette table de vérité l'équation de la sortie en fonction des entrées, de même qu'en math on résout l'équation ou le système d'équations.

- Simplifier au maximum l'équation obtenue pour minimiser le nombre de composants utilisés. Cette simplification peut s'effectuer en utilisant les simplifications algébriques vues ci-dessus ou en utilisant les tableaux de Karnaugh que nous allons voir plus loin.

C'est seulement en respectant ces 3 étapes que l'on peut être assuré d'arriver à une solution du problème CORRECTE et OPTIMALE !

Le plus simple est d'étudier un exemple concret :

Cahier des charges :

Sur un pétrolier, la cale comprend 3 soutes à pétrole (A, B et C). Elles sont remplies de façon indépendante les unes des autres. Quand le remplissage est terminé, un voyant V doit s'allumer si l'assiette du bateau est correcte, c'est à dire si le pétrole est correctement réparti dans les soutes. Des capteurs testent le remplissage complet de chaque soute.

L'assiette est correcte si les 3 soutes sont vides ou si elles sont toutes les 3 remplies ou si seule B est remplie ou si seules A et C sont remplies.

Définition des entrées du système

Capteur a : a = 1 si la soute A est remplie
Capteur b : b = 1 si la soute B est remplie
Capteur c : c = 1 si la soute C est remplie

⇨

Sortie du système

V = 1 si l'assiette est correcte

1. 1ère étape : traduction de l'énoncé dans une table de vérité

Le système comporte 3 entrées : il existe 2^3 cas possibles soit 8 cas : la table de vérité aura 8 lignes

c	b	a	V	remarques
0	0	0	1	cuves vides
0	0	1	0	déséquilibre
0	1	0	1	B seule pleine
0	1	1	0	déséquilibre
1	0	0	0	déséquilibre
1	0	1	1	A et C pleines
1	1	0	0	déséquilibre
1	1	1	1	3 cuves pleines

Rem : si l'on affecte un poids à chaque bit, on compte bien de haut en bas de 0 à 7. Dans les documents techniques relatifs aux compteurs binaires (que nous verrons plus loin) il est habituel de nommer "a" la variable de poids faible qui est à droite. Cela explique l'ordre des colonnes : c, b, a de gauche à droite.

A ce stade, on est sur que le problème est traité de façon exhaustive et correcte, c'est à dire qu'on n'a oublié aucun cas de figure et qu'on répond bien au cahier des charges (si la table est bien remplie !)

2. 2ème étape : détermination de l'équation de V en fonction de a, b et c.

On voit en étudiant la table que V = 1 si a et b et c = 0 ou si a = 0 et b = 1 et c = 0 ou si a = 1 et b = 0 et c = 1 ou si a et b et c = 1.

En remplaçant les "et" et les "ou" par leurs symboles logiques, on obtient l'équation suivante :
$V = \bar{a}.\bar{b}.\bar{c} + \bar{a}.b.\bar{c} + a.\bar{b}.c + a.b.c$

Si l'on s'arrêtait là, il faudrait 3 portes NON, 4 portes ET à 3 entrées et 1 porte OU à 4 entrées ce qui serait un peu compliqué.

3. 3ème étape : simplification de cette équation

a) Méthode algébrique

On peut factoriser les 2 1ers termes par $\bar{a}.\bar{c}$ et les 2 derniers par a.c, soit :
$V = \bar{a}.\bar{b}.\bar{c} + \bar{a}.b.\bar{c} + a.\bar{b}.c + a.b.c = (\bar{a}.\bar{c})(\bar{b} + b) + (a.c)(\bar{b} + b) = \bar{a}.\bar{c} + a.c$

En étudiant cette dernière équation, on voit que V = 1 si a = c = 0 ou si a = c = 1 c'est à dire si a = c. C'est un NON-OU exclusif et on peut donc écrire $V = \overline{a \oplus c}$

On est sur maintenant d'avoir trouvé la solution optimale car on ne peut pas simplifier plus cette équation.

Rem : dans un cas simple comme celui-ci, la solution était évidente : pour que le bateau soit équilibré, il suffit que les 2 soutes extrêmes (A et C) soient toutes les 2 vides ou pleines. Mais la solution d'un problème n'est pas toujours aussi évidente...

b) Méthode du tableau de Karnaugh

Le tableau de Karnaugh est une façon différente de présenter la table de vérité. Il utilise le code Gray dans lequel une seule variable change à la fois entre 2 valeurs consécutives.

V		b a			
		0 0	0 1	1 1	1 0
c	0				
	1				

Comme il y a 3 variables, le tableau comporte 8 cases soit 4 colonnes et 2 lignes. (l'inverse serait équivalent)

Pour les colonnes, la variable de gauche (b) correspond aux chiffres de gauche (0, 0, 1, 1) et la variable de droite (a) correspond aux chiffres de droite (0, 1, 1, 0).

L'ordre des colonnes (de gauche à droite) correspond bien au code Gray : 00, 01, 11, 10.

Rem : - on garde le même ordre que dans la table de vérité : c, b, a de gauche à droite.
- S'il y avait 4 variables, on aurait dessiné un tableau carré de 4 colonnes sur 4 lignes et on aurait respecté pour l'ordre des lignes le code Gray comme pour les colonnes.

On remplit le tableau en reportant dans les cases correspondantes les valeurs de la table de vérité, ce qui donne le résultat suivant.

V		b a			
		0 0	0 1	1 1	1 0
c	0	1	0	0	1
	1	0	1	1	0

L'astuce consiste à regrouper des cases adjacentes contenant des 1 par puissances de 2 : on peut regrouper 2 cases, 4 cases, 8 cases (dans un tableau de 16 cases) etc…

Dans le tableau ci-dessus, les deux 1 du bas sont bien adjacents : on peut les regrouper dans une "bulle" de 2 cases et on constate que, dans cette "bulle", les variables a et c ne changent pas et restent à 1, par contre la variable b passe de 0 à 1.
On oublie donc la variable b qui change et l'équation de V contiendra a.c (car a = c = 1)

Les deux 1 du haut ne sont apparemment pas dans des cases adjacentes mais il faut voir le tableau comme s'il était enroulé sur un cylindre : les colonnes de droite et de gauche sont aussi considérées comme adjacentes. Donc on peut regrouper ces deux 1 dans une seule bulle (indiquée sur la figure). Dans cette "bulle", a et c ne changent pas et restent à 0, par contre b change.
On oublie donc la variable b qui change et l'équation de V contiendra $\overline{a}.\overline{c}$ (car a = c = 0)

En faisant un OU entre les 2 "bulles", on obtient en définitive $V = \overline{a}.\overline{c} + a.c$

On a obtenu directement l'équation la plus simple possible sous la forme d'une somme de produits.

Comparaison avec la méthode algébrique

Considérons seulement les deux 1 du bas : si l'on n'avait pas regroupé les 2 cases on aurait écrit : $a.\overline{b}.c + a.b.c$ puis mis a.c en facteur soit $a.c.(b + \overline{b})$. Comme $b + \overline{b} = 1$, il reste bien a.c
La même opération avec les deux 1 du haut aurait donné $\overline{a}.\overline{c}$

On voit que le regroupement de cases adjacentes dans des "bulles" remplace automatiquement la simplification par $X + \overline{X}$ (même si X contient plusieurs autres variables).

Remarques
Dans un tableau carré de 16 cases les lignes du haut et du bas sont aussi considérées comme adjacentes. On considère donc le tableau enroulé sur un tore (un pneu si vous préférez !)

S		b a			
		0 0	0 1	1 1	1 0
d c	0 0	0	1	3	2
	0 1	4	5	7	6
	1 1	12	13	15	14
	1 0	8	9	11	10

Dans les cases sont notées les valeurs décimales correspondant aux valeurs binaires correspondantes, avec d en poids fort et a en poids faible.
Si l'on présente le tableau toujours dans le même ordre (poids fort à gauche et poids faible à droite), cette disposition permet de le remplir de façon automatique et rapide.

exemple d'un tableau de 16 cases

S		b a			
		0 0	0 1	1 1	1 0
d c	0 0	0	1	1	0
	0 1	1	0	0	1
	1 1	1	0	0	1
	1 0	0	1	1	1

On regroupe les 1 en commençant par les "bulles" les plus grandes puis on regroupe le 1 en bas à droite avec celui de gauche ou celui du dessus (cela revient au même).

On obtient donc directement : $S = \overline{c}.a + c.\overline{a} + d.\overline{c}.b$
que l'on peut encore simplifier $S = a \oplus c + d.\overline{c}.b$

c) Cas d'une table de vérité incomplète

Prenons un exemple concret : on veut concevoir un circuit tel que sa sortie S = 1 si un chiffre N **codé en BCD** est un multiple de 3. Autrement dit on veut que S = 1 si N = 3, 6 ou 9. Il faut bien 4 bits pour coder les chiffres de 0 à 9 mais le cas où N > 9 ne se produira pas et toutes les combinaisons depuis 10 (1010_2) jusqu'à 15 (1111_2) ne seront jamais présentes à l'entrée du circuit.

Si l'on cherche l'équation de S en utilisant la table de vérité, on obtient le résultat suivant :

N	D	C	B	A	S
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1

$$S = \overline{D}.\overline{C}.B.A + \overline{D}.C.B.\overline{A} + D.\overline{C}.\overline{B}.A = \overline{D}.B.(C \oplus A) + D.\overline{C}.\overline{B}.A$$

Comme on note seulement les lignes pour lesquelles S = 1, cela revient à considérer que, pour les lignes 10 à 15, S = 0 : cela n'était pas obligatoire puisque ces cas ne se produiront pas.

Si l'on utilise le tableau de Karnaugh, on peut mettre des X dans ces cases puisque le résultat est indifférent :

L'intérêt est que, pour agrandir les bulles où il y a DEJA des 1, on peut englober les cases contenant des X et ainsi simplifier encore plus l'équation de la sortie.

S		B A			
		0 0	0 1	1 1	1 0
D C	0 0	0	0	1	0
	0 1	0	0	0	1
	1 1	X	X	X	X
	1 0	0	1	X	X

En partant du tableau de Karnaugh, on obtient :

$$S = D.A + C.B.\overline{A} + \overline{C}.B.A = D.A + B(C \oplus A)$$

On constate que la 2ème expression de S est nettement plus simple que la 1ère : dans les cas où N < 10, elle donnera les mêmes résultats. Dans les cas où N > 10, elle ne donnera pas les mêmes résultats mais ces cas ne se produisant pas, cela ne nous gène pas.

Conclusion

Dans le cas des tables de vérité incomplètes, on voit bien qu'il est beaucoup plus intéressant d'utiliser les tableaux de Karnaugh que les simplifications algébriques à partir des tables de vérité : celles-ci ne nous auraient pas permis de voir que certaines cases contenant des X permettaient d'agrandir les bulles où il y a déjà des 1 et donc de simplifier l'équation de la sortie (et il n'y a aucun autre moyen de voir quelles cases sont intéressantes seulement à partir de la table de vérité).

CIRCUITS COMPLEXES

Les portes simples que nous avons vues jusqu'à maintenant permettent de construire des circuits de plus en plus complexes jusqu'aux microprocesseurs les plus puissants.
Nous allons voir quelques uns des circuits les plus courants que l'on rencontre en Architecture des Systèmes.

I. Multiplexeur

Un multiplexeur est une sorte d'aiguillage logique : il permet de sélectionner une entrée de donnée parmi n et d'envoyer la donnée présente sur cette entrée vers la sortie du circuit.

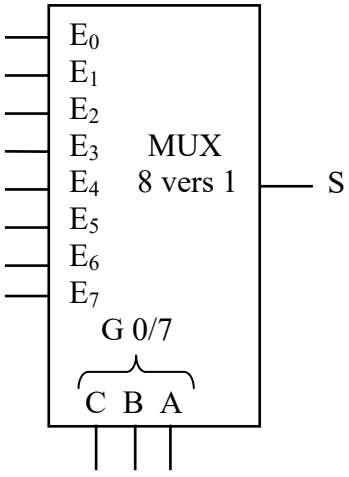
On peut faire l'analogie avec le bouton qui permet de choisir le tuner, le lecteur de CD ou le magnétophone sur une chaîne HIFI et d'envoyer le signal qui en vient vers l'ampli et les enceintes.

Pour sélectionner une entrée parmi n, il faut donner au multiplexeur le numéro de cette entrée, évidemment sous forme binaire : on appellera ce numéro **l'adresse** de l'entrée correspondante.

Par exemple avec 3 entrées d'adresses, on peut compter de 0 (000) à 7 (111) donc sélectionner une entrée parmi 8. On parlera alors de multiplexeur 8 vers 1.

De façon générale, avec n entrées d'adresses, on peut sélectionner une entrée de donnée parmi 2^n.

Schéma d'un multiplexeur 8 vers 1



Par exemple si l'on applique la valeur 6 (110) sur CBA, la donnée présente sur l'entrée E_6 sera transmise en sortie : on aura $S = E_6$.

La notation G 0/7 rappelle que ces 3 entrées ne sont pas indépendantes mais doivent être prises comme formant un nombre compris entre 0 et 7 avec C en poids fort et A en poids faible.

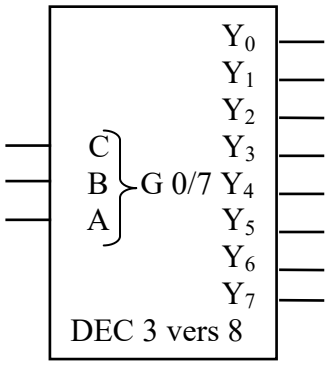
Exemple d'utilisation d'un multiplexeur : transformation d'une liaison parallèle en liaison série

A l'intérieur d'un ordinateur, les données sont véhiculées par paquets de 8 (ou 16 ou 32) sur un ensemble de 8 (ou 16 ou 32) fils qu'on appelle le **bus de données**. Pour transmettre ces données par exemple à une imprimante située loin de l'ordinateur, il revient cher d'avoir un câble qui doit comprendre au moins 8 fils sans parler de ceux permettant de gérer les échanges entre l'ordinateur et l'imprimante. Il est plus économique d'envoyer les données **les unes après les autres sur le même fil** : on applique les données aux entrées E_0 à E_7 du multiplexeur et on applique **successivement** les valeurs 0 à 7 sur les entrées d'adresses C, B et A. En sortie, on obtiendra bien sur un seul fil les données présentes sur les entrées, mais les unes après les autres.

II. Décodeur

Il en existe différentes sortes qui permettent de passer d'un code (binaire, décimal, hexa, Gray …) à un autre : nous allons en étudier un exemple simple.

Il doit comporter 3 entrées (C, B et A) et 2^3 soit 8 sorties (Y_0 à Y_7) : si l'on applique sur les entrées C, B, A un nombre binaire i sur 3 bits (donc compris entre 0 et 7), la sortie correspondante Y_i doit être égale à 1, les autres sorties restant à 0.



Par exemple si l'on applique la valeur 4 (100) sur CBA, la sortie Y_4 passe à 1, les autres restant à 0.

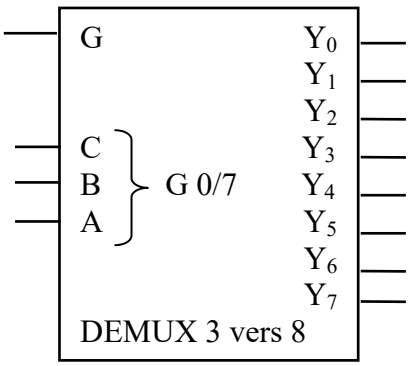
Rem :
On peut bien sur généraliser le nombre d'entrées : à n entrées correspondront 2^n sorties et on aura un décodeur n vers 2^n.

Exemple d'utilisation d'un décodeur de ce type : dans un ordinateur, les données sont stockées dans plusieurs "puces" mémoire et un décodeur permet au microprocesseur de sélectionner la bonne puce en fonction de l'adresse à laquelle on veut lire une donnée (nous en verrons un exemple en TD).

III. Démultiplexeur

C'est presque le même circuit que ci-dessus mais auquel on rajoute une entrée dite **entrée de validation**. Elle est souvent notée G et fonctionne de la façon suivante.

Si elle est inactive (par exemple à 0) le démultiplexeur ne fonctionne pas : ses sorties Y_i restent à 0 quel que soit l'état des entrées A, B et C. Si elle est activée, le démultiplexeur fonctionne comme le décodeur vu ci-dessus.



Rem :
- *On peut comparer cette entrée au bouton marche/arrêt qui fait fonctionner ou non un circuit.*
- *Les documents techniques précisent toujours l'état de G nécessaire pour que le circuit fonctionne.*
- *En fait, il n'existe qu'un seul type de circuit appelé décodeur/démultiplexeur et on utilise l'entrée G en fonction des besoins.*

Exemple d'utilisation d'un démultiplexeur : il fait le contraire d'un multiplexeur ! (c'est même pour cela qu'on l'appelle … démultiplexeur)

Dans l'exemple donné plus haut de la transmission série, il sera utilisé pour "remettre en parallèle" les données arrivant en série depuis l'émetteur : il suffit de relier son entrée G à la sortie S du multiplexeur et de mettre au **même instant les mêmes valeurs** sur les entrées C, B et A du multiplexeur et du démultiplexeur. Il faudra de plus "mémoriser" les données présentes sur les sorties Y_i pendant un cycle complet pour les retrouver présentes **toutes en même temps**, c'est à dire en parallèle.

LOGIQUE SEQUENTIELLE : LES BASCULES

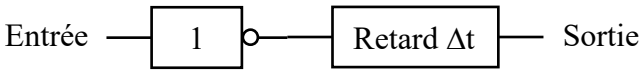
I. Introduction

1. Temps de propagation

Jusqu'ici, nous avons travaillé en logique combinatoire : si l'on connaît l'état des entrées d'un système à un instant donné, on peut déterminer l'état de la (ou des) sortie(s) à cet instant.

De plus, nous avons toujours négligé le **temps de propagation** de l'information entre l'entrée et la sortie d'un système. Les composants utilisés pour réaliser des portes logiques (diodes, transistors) ne réagissent pas instantanément aux changements d'état des entrées. Par exemple, dans un porte NON, il s'écoule un certain temps (depuis une dizaine de nanosecondes jusqu'à nettement moins pour les composants récents) entre le moment où l'entrée passe de 0 à 1 et celui où la sortie passe de 1 à 0.

On peut représenter ce délai par le schéma suivant :



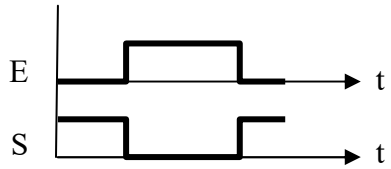
porte idéale (sans retard)

2. Chronogramme

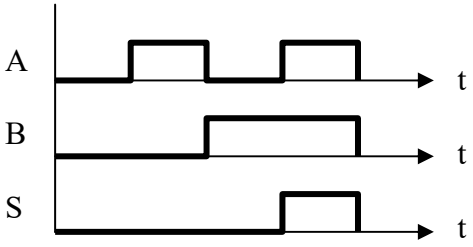
Pour représenter une fonction logique, nous avons vu les équations logiques, les tables de vérité et les tableaux de Karnaugh. Ces représentations sont suffisantes en logique combinatoire mais n'intègrent pas la notion de temps et ne permettent donc pas de visualiser facilement des systèmes où le temps intervient de façon essentielle.

C'est pourquoi on adopte plutôt les chronogrammes : ils consistent à dessiner un graphique sur lequel le temps sera représenté en abscisse et le niveau logique (0 ou 1) en ordonnée.
(avec par convention : 0 en bas et 1 … en haut !)

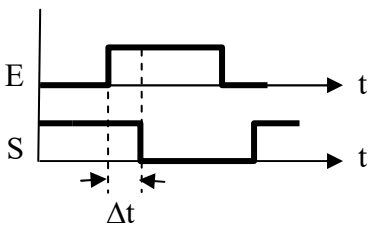
Chronogramme d'une porte NON : $S = \overline{E}$

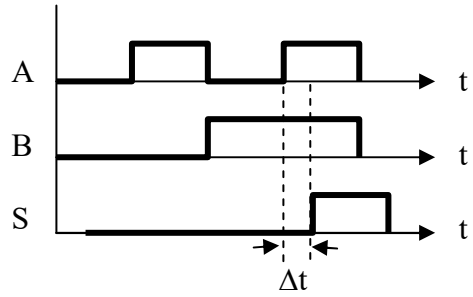


Chronogramme d'une porte ET : S = A.B



On peut faire apparaître le temps de propagation de ces portes :

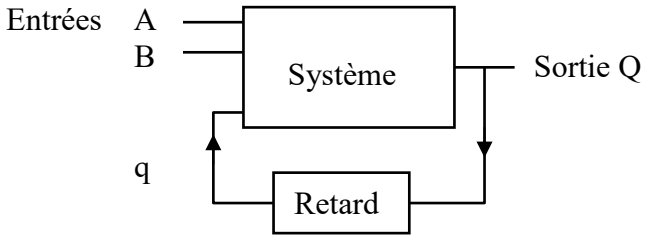




3. Logique séquentielle

On passe dans le domaine de la logique séquentielle quand le temps intervient, c'est à dire quand il faut connaître l'état antérieur de la sortie d'un système (ainsi bien sur que l'état des entrées) pour déterminer son état à un instant donné. On peut dire que le système "garde en mémoire" l'état antérieur de la sortie pour déterminer son état à un instant donné.

On peut représenter un système séquentiel à 2 entrées par le schéma suivant dans lequel on voit apparaître le "rebouclage" de la sortie Q du système sur son entrée.

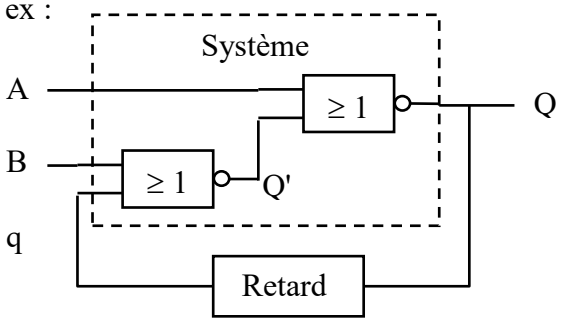


On note Q(t) = f(A, B, Q(t-1))

Pour simplifier l'écriture, on adoptera la notation suivante : Q(t) = Q et Q(t-1) = q

On pourra donc écrire : Q = f(A, B, q)

ex :



L'équation de Q est donc : $Q = \overline{A + Q'} = \overline{A + \overline{B + q}}$

On constate que Q dépend bien de A, de B et aussi de q, c'est à dire de l'état antérieur de la sortie Q.

En particulier si A = B = 0, on obtient Q = q
La sortie Q "conserve" son état antérieur :
c'est la notion de "mémoire" …essentielle en informatique !

Rem : l'appellation Q est usuelle pour les sorties de systèmes séquentiels.

Si l'on redessine différemment ce circuit en ne faisant pas apparaître explicitement le circuit de retard mais seulement implicitement à travers une différence de notation (q au lieu de Q), on obtient le schéma d'un circuit qu'on appelle bascule RS asynchrone.

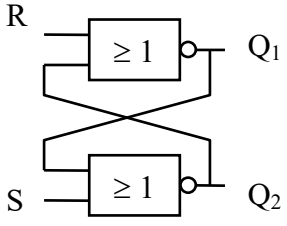
II. Les bascules RS asynchrones

Rem : une bascule est asynchrone quand ses sorties réagissent immédiatement aux changements d'état des entrées sans attendre l'activation d'une entrée auxiliaire dite entrée d'horloge.

1. Bascule RS à base de NOR

L'équation de Q_1 est : $Q_1 = \overline{R + Q_2} = \overline{R + \overline{S + q_1}} = \overline{R}.(S + q_1)$

et celle de Q_2 est : $Q_2 = \overline{S + Q_1} = \overline{S + \overline{R + q_2}} = \overline{S}.(R + q_2)$



On peut donc remplir la table de vérité de Q_1 et de Q_2 :

R	S	Q_1	remarques	Q_2	remarques
0	0	q_1	état mémoire	q_2	état mémoire
0	1	1	mise à 1 de Q_1	0	mise à 0 de Q_2
1	0	0	mise à 0 de Q_1	1	mise à 1 de Q_2
1	1	0	0*	0	0*

Cette table de vérité demande quelques précisions **essentielles** pour bien comprendre la suite.

- 1ère ligne : R = S = 0 ⇒ Q_1 et Q_2 sont dans **l'état mémoire** : les sorties **restent** dans l'état où elles étaient juste avant le retour à 0 de R et de S.

- 2ème ligne : R = 0 ; S = 1 ⇒ $Q_1 = 1$ et $Q_2 = 0$ **quel que soit l'état antérieur de ces sorties**. Si l'on considère Q_1 comme la sortie principale de la bascule RS, le fait d'activer S (de positionner cette entrée à 1) et pas R force Q_1 à 1 : l'entrée S est appelée **entrée de mise à 1** (en anglais **Set** d'où son initiale).

- 3ème ligne : R = 1 ; S = 0 ⇒ $Q_1 = 0$ et $Q_2 = 1$ **quel que soit l'état antérieur de ces sorties**. Le fait d'activer R (de positionner cette entrée à 1) et pas S force Q_1 à 0 : l'entrée R est appelée **entrée de mise à 0** (en anglais **Reset** d'où son initiale). On l'appelle aussi RAZ (pour Remise A Zéro).

- 4ème ligne : R = S = 1 ⇒ $Q_1 = Q_2 = 0$ **quel que soit l'état antérieur de ces sorties**. Si l'on fait en même temps une demande de mise à 1 en activant S et une demande de mise à 0 en activant R, la sortie principale Q_1 passe (ou reste) à 0.

  Pour cette raison une bascule RS à base de NOR est appelée bascule **à arrêt prioritaire** (par analogie avec des machines commandées par un interrupteur Marche/Arrêt, on associe le niveau 1 à la position Marche et le niveau 0 à la position Arrêt).

Mais ce dernier cas est particulier d'où l'étoile à coté du 0. En effet dans les 3 premiers cas, on a toujours $Q_1 = \overline{Q_2}$. C'est évident pour les lignes 2 et 3 mais c'est aussi le cas pour la 1ère ligne et voici pourquoi.

A l'intérieur des portes NOR, les transistors et autres composants ne sont jamais **totalement** identiques les uns avec les autres. En particulier certains réagiront (changeront d'état) légèrement plus vite que d'autres.
Partons de l'hypothèse qu'un 0 apparaît **d'abord** sur la sortie Q_1. Dans ce cas, la porte NOR du bas voit des 0 sur ses **2** entrées. La sortie Q_2 se positionne donc à 1, ce qui maintient la sortie Q_1 à 0. Le système est stable dans cette position et les 2 sorties sont bien dans des états complémentaires.
Mais si nous étions partis de l'hypothèse inverse (0 sur Q_2 en premier) nous serions arrivés à la même conclusion : Q_1 et Q_2 sont bien dans des états complémentaires (inverses du cas précédent).

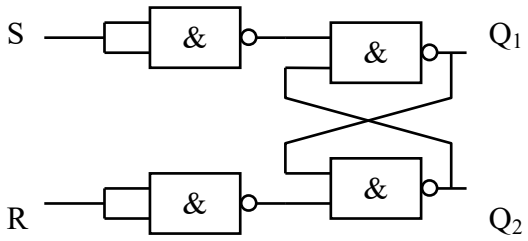
Autrement dit, $Q_1 = \overline{Q_2}$ **sauf si R = S = 1**

Imaginons maintenant que les entrées R et S repassent **en même temps** à 0. Juste avant de repasser à 0, elles étaient à … 1(bravo !), et on avait $Q_1 = Q_2 = 0$. Juste après le passage à 0, on doit avoir $Q_1 = \overline{Q_2}$. Laquelle des 2 passe à 1 ? **On n'en sait rien** et ce n'est pas acceptable pour un système informatique qui doit être totalement prévisible et ne doit surtout pas dépendre de la rapidité d'un composant par rapport à un autre.

Cette explication (un peu longue …) justifie le fait que le dernier état est appelé **<u>état interdit</u>** et qu'on l'évite soigneusement !

Si l'on évite cet état particulier, on a toujours $Q_1 = \overline{Q_2}$: on appellera Q la sortie Q_1 et $\overline{Q}$ la sortie Q_2

2. Bascule RS à base de NAND



L'équation de Q_1 est : $Q_1 = \overline{\overline{S}.Q_2} = \overline{\overline{S}.\overline{R}.q_1} = S + \overline{R}.q_1$

et celle de Q_2 est : $Q_2 = \overline{\overline{R}.Q_1} = \overline{\overline{R}.\overline{S}.q_2} = R + \overline{S}.q_2$

On peut donc remplir la table de vérité de Q_1 et de Q_2 :

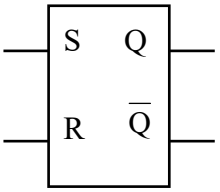
R	S	Q_1	remarques	Q_2	remarques
0	0	q_1	état mémoire	q_2	état mémoire
0	1	1	mise à 1 de Q_1	0	mise à 0 de Q_2
1	0	0	mise à 0 de Q_1	1	mise à 1 de Q_2
1	1	1	1*	1	1*

Les 3 premières lignes de la table de vérité de cette bascule sont identiques à celles correspondant à la bascule RS à base de NOR et ne demandent aucun commentaire supplémentaire.

Seule la dernière ligne diffère : R = S = 1 : $Q_1 = Q_2 = 1$ **quel que soit l'état antérieur de ces sorties** Pour cette raison une bascule RS à base de NAND est appelée bascule **à marche prioritaire.**

Si l'on évite ce dernier état (et on a vu qu'il valait mieux l'éviter !), ces 2 bascules sont rigoureusement équivalentes et par la suite on ne les distinguera plus.

Symbole d'une bascule RS asynchrone

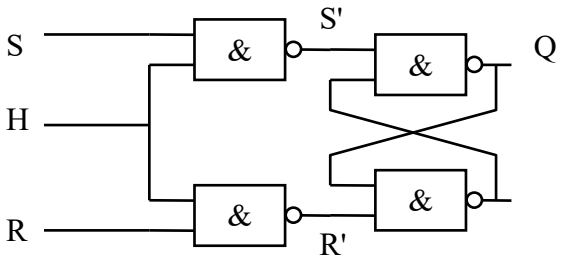


III. Les bascules RS synchrones

Une bascule est synchrone quand ses sorties ne changent d'état que si un signal supplémentaire est appliqué sur une entrée, dite **entrée d'horloge**, selon certaines modalités que nous allons détailler.

1. Synchronisation sur niveau (ou état)

On voit dans le schéma ci-contre qu'au lieu d'utiliser les 2 NAND de gauche en inverseurs comme dans le schéma précédent, on relie une de leurs entrées qui devient l'entrée d'horloge.



On pourrait calculer l'équation de Q et établir sa table de vérité, mais il est plus simple de raisonner par analogie avec la bascule précédente.

- Tant que H = 0, S' = R' = 1 : la bascule est dans l'état mémoire : on peut modifier l'état des entrées S et R mais cela n'a aucune influence sur Q.

- Tant que H = 1, $S' = \overline{S}$ et $R' = \overline{R}$: on retrouve le schéma précédent : la bascule synchrone est rigoureusement identique à la bascule asynchrone.

Mais ce type de synchronisation a quelques inconvénients : la bascule est "sensible" aux entrées pendant toute la durée de l'état 1 de l'horloge. Si, pendant que H = 1, des parasites apparaissent sur les entrées S ou R, ils peuvent entraîner des changements d'état imprévus sur la sortie Q.

2. Synchronisation sur front

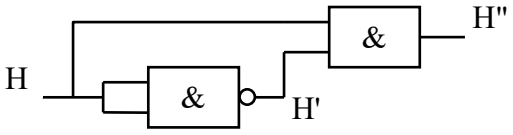
Afin de minimiser au maximum la durée de cet état sensible, on s'arrange pour que la bascule reste dans son état mémoire **sauf pendant un bref instant, juste au moment où l'entrée d'horloge PASSE de 0 à 1 (ou de 1 à 0).**

La bascule est dite synchronisée **sur front** (en anglais edge triggered)

Le front peut être montant (passage de H de 0 à 1) ou descendant (passage de H de 1 à 0)

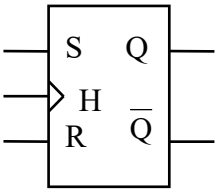
Pour obtenir ce résultat, on intercale entre l'entrée d'horloge et les NAND un circuit supplémentaire.

exemple de circuit permettant d'obtenir une synchronisation sur front montant :

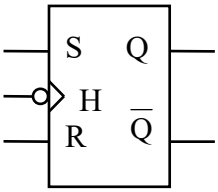


Rem : si les changements d'état des entrées ont lieu pendant que H = 0, les bascules synchronisées sur front montant se comportent comme des bascules synchronisées sur niveau.

Symbole d'une bascule RS synchronisée sur front montant



sur front descendant



Ce type de synchronisation a encore des inconvénients : on a déjà évoqué le fait que les composants n'ont pas un temps de réaction nul : dans un système informatique un peu complexe, certaines bascules en commandent d'autres, elles n'ont pas toutes exactement les mêmes temps de réaction et ces différences peuvent générer des dysfonctionnements pour le moins gênants !

3. Synchronisation sur impulsion (bascule Maître-Esclave)

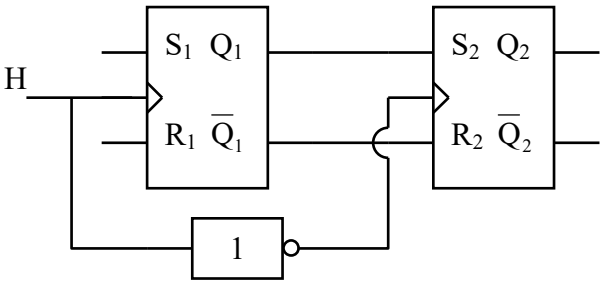
Pour éviter ces problèmes on construit des bascules qui fonctionnent de la façon suivante :

Au moment du front montant de l'horloge, la bascule "mémorise" l'état des entrées R et S présents à cet instant, mais sa sortie Q ne change pas d'état. La bascule garde en mémoire l'état de R et de S jusqu'au front descendant suivant et là elle effectue (éventuellement) les changements d'état prévus au moment du front montant.

Autrement dit, il faut 2 fronts pour effectuer une action : le 1er (le front montant) pour "mémoriser" et le 2ème (le front descendant) pour que la sortie Q change.

Cette succession de 2 fronts s'appelle une **impulsion** (positive si le 1er front est montant et le 2ème descendant et négative dans le cas contraire).

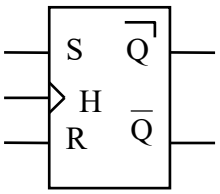
Pour obtenir ce résultat, on utilise 2 bascules RS dont la 1ère est appelée Maître (celles qui mémorise), et la 2ème Esclave (celle qui exécute les ordres du Maître et commande les sorties), l'horloge étant inversée entre les 2 bascules



Rem : si les changements d'état des entrées ont lieu pendant que H = 0, les bascules synchronisées sur impulsion se comportent comme des bascules synchronisées sur front descendant puisque Q ne peut changer d'état que sur le front descendant de H.

Symbole d'une bascule RS synchronisée sur impulsion positive

Le triangle à coté de l'entrée d'horloge rappelle que la mémorisation se fait sur front montant et le signe à coté de la sortie Q symbolise un front descendant : il rappelle que la sortie Q ne peut basculer que sur le front descendant de H.

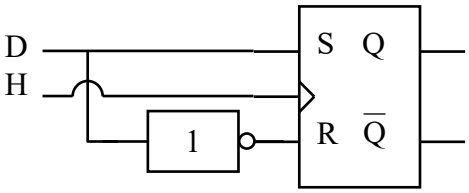


IV. Les bascules D

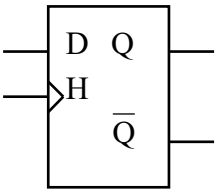
Remarque préliminaire : en dehors des bascules RS qui peuvent être synchrones ou asynchrones, toutes les bascules que nous allons voir par la suite sont forcément des bascules synchrones.

On obtient une bascule D en rajoutant un inverseur entre S et R (voir le schéma)
L'équation d'une bascule D est donc très simple : Q = D : la sortie Q recopie l'état de l'entrée D en fonction du mode de synchronisation.

Schéma



Symbole



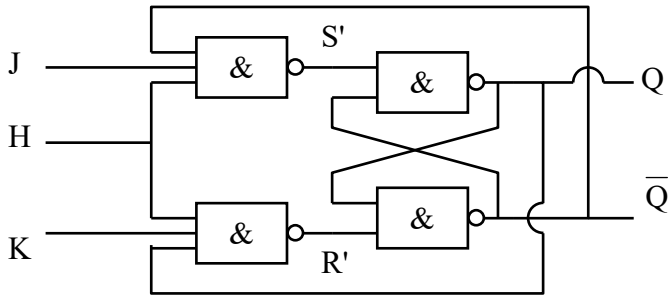
Il est évident sur ce schéma qu'une bascule D asynchrone n'aurait pas de sens : ce serait une simple porte OUI (Q = D) sans aucune mémoire … et un simple fil la remplacerait avantageusement !

Les bascules D existent avec les 3 modes de synchronisation vus ci-dessus.

V. Les bascules JK

On a vu que, pour les bascules RS, l'état R = S = 1 est interdit. On va "récupérer" ce cas pour obtenir une commande supplémentaire.

Pour cela, on remplace les NAND de gauche par des NAND à 3 entrées et on relie " l'ancienne" entrée S à $\overline{Q}$ et " l'ancienne" entrée R à Q.



Admettons de plus que la bascule est synchronisée sur front montant (il faudrait rajouter sur l'entrée H le circuit vu au paragraphe III. 2., non représenté ici pour ne pas alourdir le schéma)

- J = K = 0 $\Rightarrow$ S' = R' = 1 : la bascule est dans l'état mémoire : Q = q
- J = 1 ; K = 0 $\Rightarrow$ J "joue le rôle" de S et K celui de R : $\Rightarrow$ Q = 1 après un front montant sur H.
- J = 0 ; K = 1 $\Rightarrow$ pour la même raison que ci-dessus Q = 0 après un front montant sur H.
- J = K = 1 $\Rightarrow$ on peut "oublier" ces 2 entrées car des 1 à l'entrées d'un NAND ne modifient pas sa sortie (X.1 = X). Supposons que Q = 1 avant le front montant d'horloge. Au moment où celui-ci arrive, l'entrée des NAND de gauche passe à 1 pendant un bref instant. Donc R' (= $\overline{Q.H}$) passe à 0.

  D'où $\overline{Q}(=\overline{R'.Q})$ passe à 1et Q $(=\overline{S'.\overline{Q}})$ passe à 0. On voit qu'au moment du front d'horloge Q change d'état.

Un raisonnement analogue aurait prouvé la même chose en partant de Q = 0

En définitive J = K = 1 $\Rightarrow$ $Q = \overline{q}$: la sortie Q bascule systématiquement sur le front montant

On peut regrouper tous ces cas de figure dans une table de vérité

J	K	Q	remarques
0	0	q	état mémoire
0	1	0	mise à 0
1	0	1	mise à 1
1	1	$\overline{q}$	basculement

Les 3 premières lignes correspondent bien à une bascule RS mais la dernière ligne introduit un mode supplémentaire qui sera très utilisé pour réaliser les compteurs synchrones que nous verrons plus loin.

On peut déduire de cette table l'équation de Q :

$$Q = \overline{J}.\overline{K}.q + J.\overline{K} + J.K.\overline{q} = \overline{J}.\overline{K}.q + J.(\overline{K} + K.\overline{q}) = \overline{J}.\overline{K}.q + J.(\overline{K} + \overline{q}) = \overline{J}.\overline{K}.q + J.\overline{K} + J.\overline{q}$$
$$= \overline{K}(\overline{J}.q + J) + J.\overline{q} = \overline{K}(q + J) + J.\overline{q} = \overline{K}.q + \overline{K}.J + J.\overline{q} = \overline{K}.q + J.\overline{q}$$

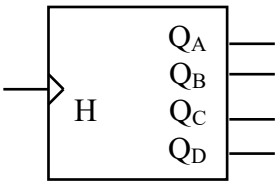
L'équation d'une bascule JK est donc $\boxed{Q = J.\overline{q} + \overline{K}.q}$

LES COMPTEURS

I. Introduction

Un compteur est un circuit qui …compte des impulsions et qui affiche sur ses sorties le nombre d'impulsions qu'il a reçues depuis le début du comptage (en binaire évidemment).

On le représente par le schéma suivant :
(compteur sur 4 bits synchronisé sur front montant)
Q_A est bien sur la sortie de poids faible et Q_D celle de poids fort.

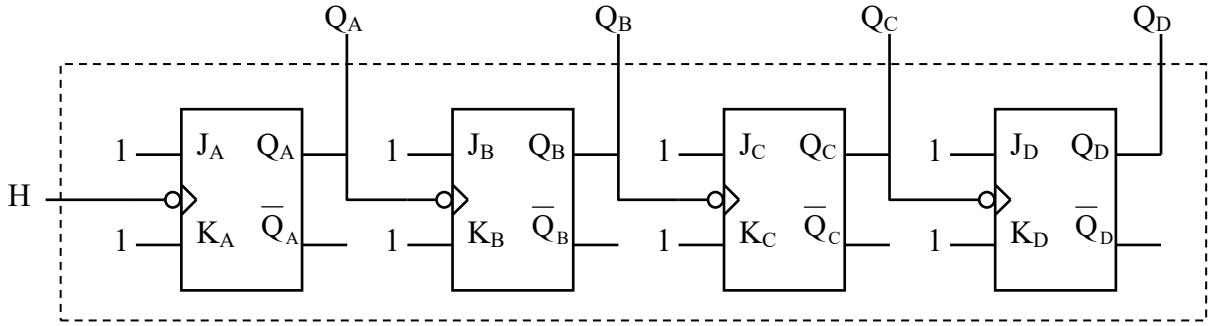


Il existe 2 types de compteurs : les compteurs asynchrones et les compteurs synchrones
(les termes "synchrone" ou "asynchrone" n'ont pas la même signification que pour les bascules et les 2 types de compteurs sont réalisés avec des bascules synchrones)

II. Les compteurs asynchrones

Ils sont réalisés avec des bascules montées en diviseurs de fréquence par 2 (avec des bascules JK, il suffit de mettre J et K à 1)

1. Modulo 2^n (sur 4 bits)



Explication : pour compter en binaire naturel il faut que le bit de rang i + 1 change d'état quand le bit de rang i repasse à 0 : si l'on utilise des bascules synchronisées sur front descendant, il suffit de relier les sorties Q_i aux entrées d'horloge des bascules de rang i + 1, ce qui est réalisé sur le schéma ci-dessus.

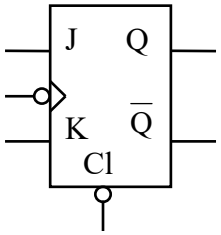
Le compteur représenté ici comptera donc de 0 (0000) à 15 (1111), le retour à 0 se faisant automatiquement à la $16^{ème}$ impulsion d'horloge.

En généralisant, on voit qu'avec n bascules, on obtient des compteurs modulo 2^n.

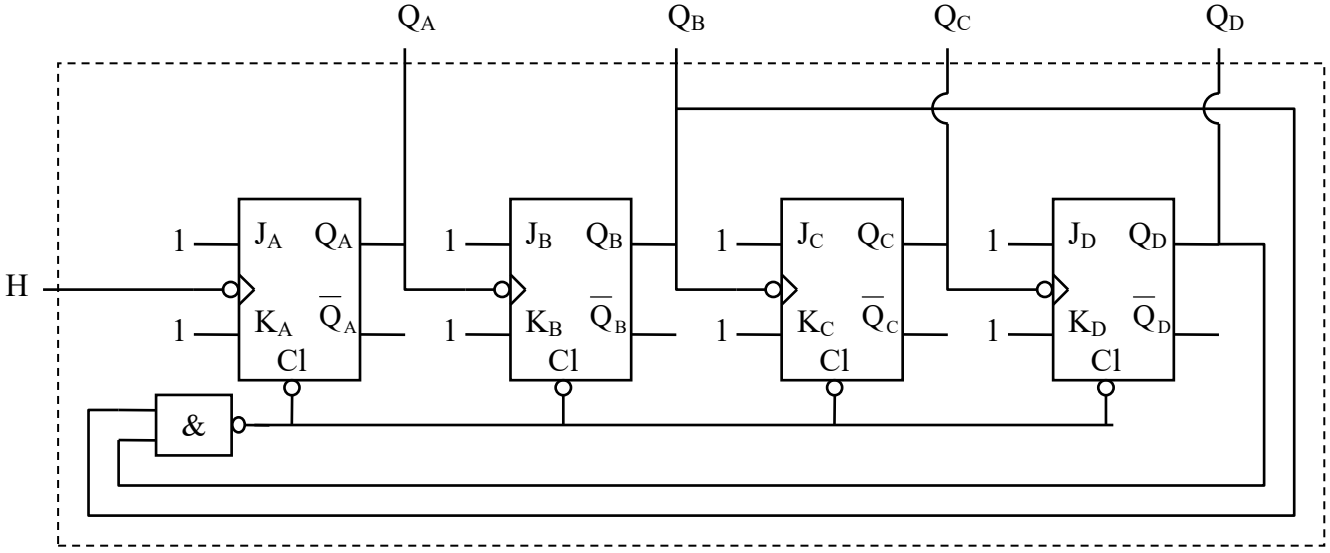
2. Modulo $\neq 2^n$

Pour revenir à 0 sur une autre valeur qu'une puissance de 2, il faut détecter cette valeur et remettre immédiatement le compteur à 0 : On utilisera l'entrée de remise à 0 asynchrone que possèdent toutes les bascules. Elle fonctionne de la façon suivante : dès que l'on applique l'état actif sur cette entrée, la sortie Q de la bascule passe à 0, indépendamment de l'état de l'horloge et des autres entrées de commande (D ou J, K). Elle est notée R(eset) ou Cl(ear) et elle est souvent active à l'état bas.

Schéma d'une bascule JK, synchronisée sur front descendant et munie d'une entrée de remise à 0 active à l'état bas.



Par exemple pour réaliser un compteur asynchrone décimal, c'est à dire qui compte de 0 à 9 et qui revient à 0 à la $10^{ème}$ impulsion, il faut détecter la valeur 10_{10} soit 1010_2 et activer les entrées Clear des bascules par un niveau bas. La remise à 0 doit donc être activée dès que $Q_D = Q_B = 1$ et $Q_C = Q_A = 0$ mais comme on compte par ordre croissant, c'est la $1^{ère}$ fois que Q_D et Q_B sont tous les deux à 1 en même temps : il suffit donc de détecter le passage à 1 de ces 2 sorties et d'envoyer un 0 sur les entrées Clear des 4 bascules, ce qui se fait avec un simple porte NAND selon le schéma suivant.



Rem : par mesure de précaution, on remet à 0 toutes les bascules, même les bascules A et C qui y sont déjà.

III. Les décompteurs asynchrones

1. Modulo 2^n (sur 4 bits)

Pour décompter en binaire naturel il faut que le bit de rang i + 1 change d'état quand le bit de rang i passe à 1 : si l'on utilise des bascules synchronisées sur front descendant, il suffit de relier les sorties $\overline{Q_i}$ aux entrées d'horloge des bascules de rang i + 1. Le schéma est donc analogue à celui des compteurs modulo 2^n à la remarque ci-dessus près.

2. Modulo ≠ 2^n

Le principe est analogue à celui des compteurs à quelques détails près : pour réaliser un décompteur modulo n, on doit obtenir la séquence suivante : 3, 2, 1, 0, n-1, n-2, …., 3, 2, …
La valeur 0 doit bien exister et rester stable tant que la $n^{ième}$ impulsion n'est pas arrivée : si le modulo n était une puissance de 2, la valeur suivant 0 serait n-1 c'est à dire que toutes les sorties des bascules seraient à 1. C'est cette valeur-là qu'il faut remplacer par la valeur n-1.

Par exemple pour réaliser un décompteur asynchrone décimal, c'est à dire qui décompte de 9 à 0 et qui revient à 9 à la $10^{ème}$ impulsion, il faut détecter la valeur 15_{10} soit 1111_2 et la remplacer par 9_{10} soit 1001_2.
La méthode la plus simple serait de détecter la valeur 15_{10} et de remettre les sorties Q_C et Q_B à 0 en utilisant un simple NAND à 4 entrées. Dans la mesure où la valeur 10_{10} soit 1010_2 n'apparaît pas dans le cycle normal, il suffit de détecter celle-ci (soit $Q_D = Q_B = 1$) et de remettre Q_C et Q_B à 0 en utilisant un simple NAND à 2 entrées au lieu de 4.

Pour plus de sécurité, on préfère repositionner toutes les sorties dans l'état correct c'est à dire 1001.

On doit donc utiliser des bascules disposant, en plus de l'entrée de remise à 0 (entrée Clear) d'une entrée de remise à 1 fonctionnant de la même façon que l'entrée Clear : dès que l'on applique l'état actif sur cette entrée, la sortie Q de la bascule passe à 1, indépendamment de l'état de l'horloge et des autres entrées de commande (D ou J, K). Elle est notée Pr(eset) et elle est souvent active à l'état bas.

Schéma d'une bascule JK, synchronisée sur front descendant et munie d'une entrée de remise à 0 et d'une entrée de remise à 1 actives à l'état bas.

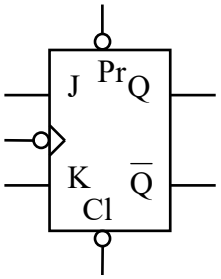
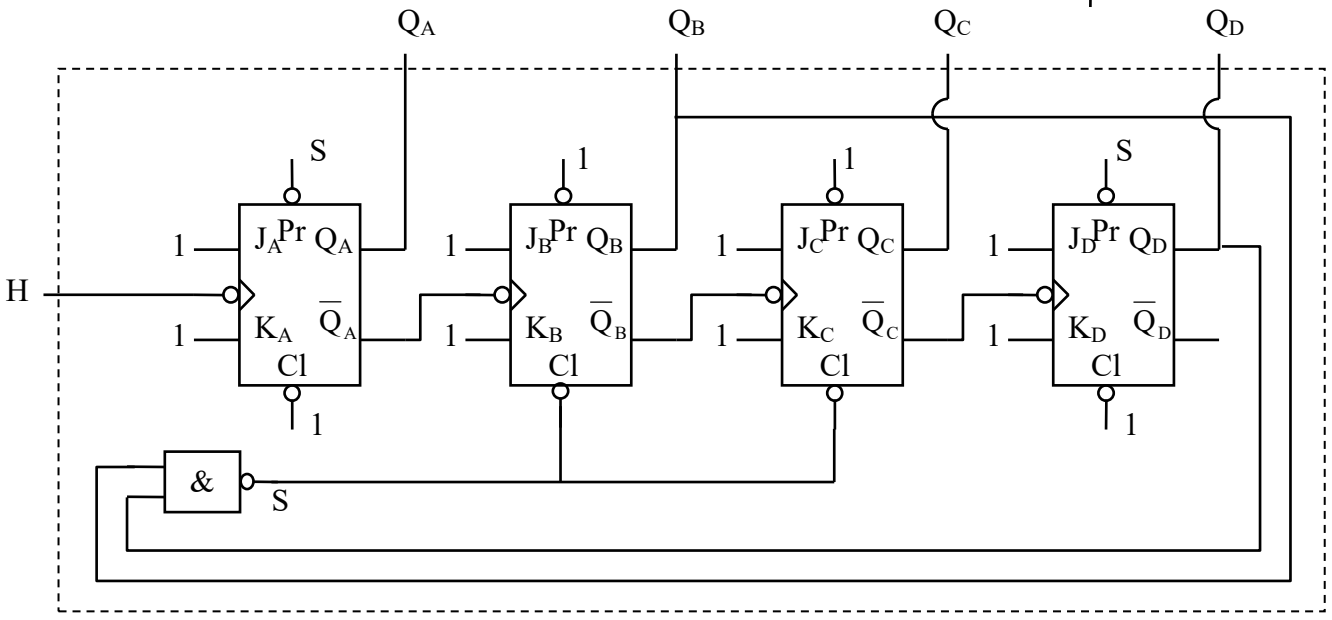


Schéma d'un décompteur asynchrone décimal :



Rem : les entrées Cl et Pr inutilisées sont positionnées dans leur état inactif c'est à dire 1.
Les entrées Pr des bascules A et D sont reliées à la sortie S du NAND mais les fils de liaison ne sont pas représentés pour ne pas alourdir le schéma.

IV. Les compteurs synchrones

Les compteurs asynchrones ont un certain nombre d'inconvénients :

- On ne peut compter ou décompter que dans l'ordre binaire pur : impossible par exemple de construire un compteur en code Gray.

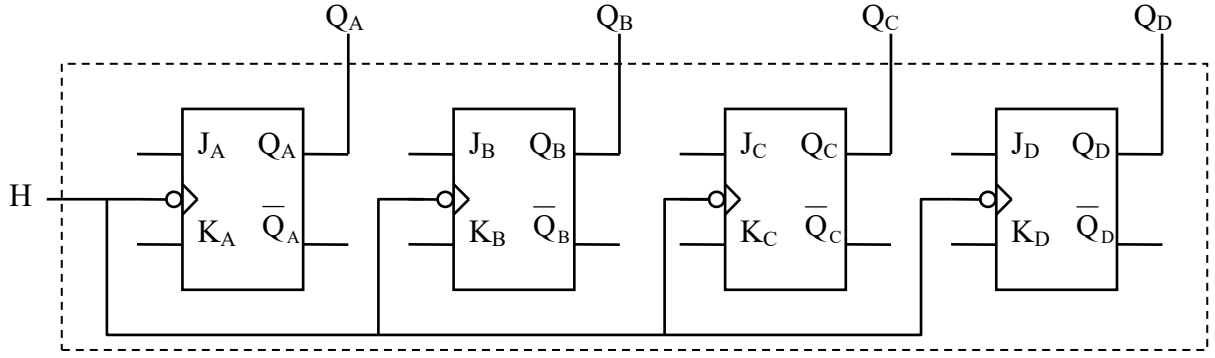
- La valeur du modulo apparaît un bref instant, pendant le temps que mettent les bascules pour réagir à l'ordre de remise à 0 ou à 1 (cet inconvénient peut toutefois être supprimé au prix d'une plus grande complexité dans la remise à 0 du compteur).

- La fréquence maximum de comptage est limitée par les temps de retard des bascules qui **s'ajoutent** : si chaque bascule a un temps de retard de 10 ns, le passage de 0111 à 1000 ne sera effectif qu'au bout de 40ns, le temps que la dernière bascule (D) ait réagi.

- Toutes les bascules doivent disposer d'entrées de forçage Reset (et Preset pour les décompteurs)

Pour toutes ces raisons, on préfère utiliser des compteurs dits synchrones : dans ceux-ci, les entrées d'horloge des bascules sont toutes reliées ensemble contrairement aux compteurs asynchrones dans lesquels la sortie d'une bascule commandait l'entrée d'horloge de la suivante.

Pour un compteur de modulo ≤ 16, cela donne le schéma suivant :



Le problème va donc consister à relier correctement les entrées J et K de chaque bascule aux sorties Q des autres bascules pour obtenir le cycle voulu.

On va utiliser l'équation d'une bascule JK qui donne la valeur de la sortie Q en fonction de son état antérieur noté q et des états de J et de K au moment du front d'horloge, soit : $Q = J.\overline{q} + \overline{K}.q$

Le plus simple est de partir d'un exemple concret, celui d'un compteur modulo 5 qui nécessitera 3 bascules C, B et A. On doit donc obtenir le cycle 0, 1,2, 3, 4, 0, 1 … : on commence par remplir la table de vérité correspondant au cycle voulu (voir page suivante).

Puis il faut remplir les tableaux de Karnaugh de chaque sortie Q **en mettant dans la case correspondant à l'état du compteur à un instant t l'état que l'on veut à l'instant t + 1**, c'est à dire après l'apparition de la $n^{ième}$ impulsion.

	N_{10}	Q_C	Q_B	Q_A
état initial	0	0	0	0
après la 1ère impulsion	1	0	0	1
après la 2ème impulsion	2	0	1	0
après la 3ème impulsion	3	0	1	1
après la 4ème impulsion	4	1	0	0
après la 5ème impulsion	0	0	0	0

ce "1" là correspond à la valeur de Q_A **après** la 1ère impulsion, c'est à dire **après** l'état 0_{10} : il doit donc être reporté dans la case 000_2 du tableau de Karnaugh correspondant à Q_A.

ce "0" là correspond à la valeur de Q_C **après** la 5ème impulsion, c'est à dire **après** l'état 4_{10} : il doit donc être reporté dans la case 100_2 du tableau de Karnaugh correspondant à Q_C.

Q_A	q_Bq_A				
		00	01	11	10
q_C	0	1	0	0	1
	1	0	X	X	X

Q_B	q_Bq_A				
		00	01	11	10
q_C	0	0	1	0	1
	1	0	X	X	X

Q_C	q_Bq_A				
		00	01	11	10
q_C	0	0	0	1	0
	1	0	X	X	X

On en tire les équations des sorties Q en utilisant **éventuellement** les valeurs indéterminées X :

$Q_A = \overline{q_C}.\overline{q_A}$ | $Q_B = \overline{q_B}.q_A + q_B.\overline{q_A}$ | $Q_C = q_B.q_A$

On identifie chaque équation à l'équation type d'une bascule JK : $Q = J.\overline{q} + \overline{K}.q$: le coefficient de $\overline{q}$ doit être égal à J et celui de q doit être égal à $\overline{K}$.

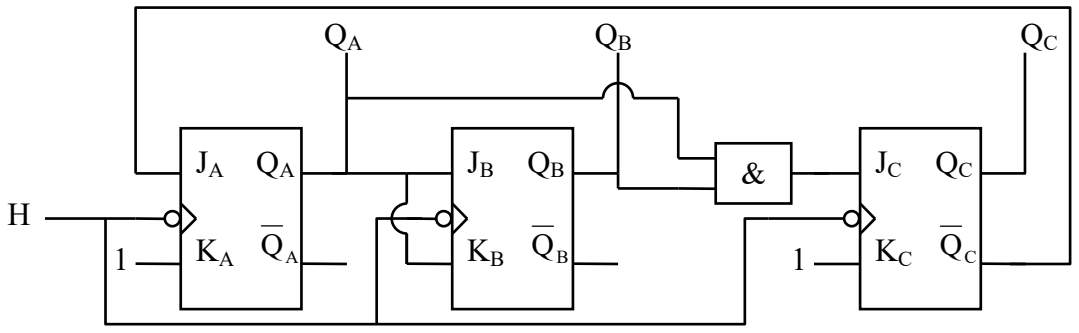
- Bascule A : le terme en q_A n'existe pas : on peut le rajouter sans modifier l'équation de Q_A en la réécrivant sous la forme : $Q_A = \overline{q_C}.\overline{q_A} + 0.q_A$: il vient donc : $J_A = \overline{q_C}$ et $K_A = 1$

- Bascule B : on obtient directement : $J_B = q_A$ et $K_B = q_A$.

- Bascule C : le terme q_C n'apparaît pas dans l'équation de Q_C : on ne peut pas identifier les coefficients. Dans ce cas on n'agrandit pas la "bulle" : on ne prend pas le X.
  .On obtient donc : $Q_C = \overline{q_C}.q_B.q_A$ ou, comme pour la bascule A : $Q_C = \overline{q_C}.q_B.q_A + 0.q_C$
  soit en définitive $J_C = q_B.q_A$ et $K_C = 1$.

La règle est donc : on agrandit les "bulles" comme d'habitude sauf si cet agrandissement fait disparaître la variable minuscule de même indice que la variable majuscule.

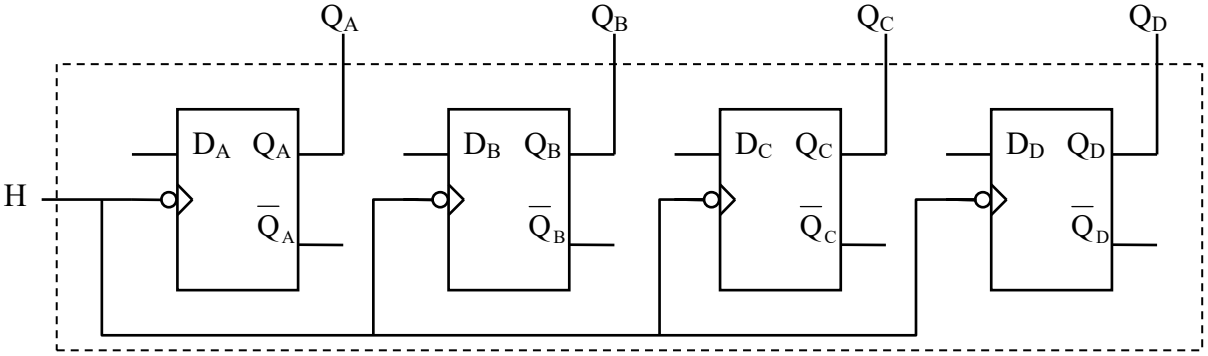
Résumé : $J_A = \overline{q_C}$ et $K_A = 1$; $J_B = K_B = q_A$; $J_C = q_B.q_A$ et $K_C = 1$ ce qui donne le schéma suivant :



LES REGISTRES A DECALAGE

I. Présentation

Un registre est un ensemble de cellules mémoire pouvant de stocker des informations binaires. Une bascule D permettant de mémoriser **un** bit, on obtient un registre 4 bits en juxtaposant 4 bascules D identiques, les entrées d'horloge étant toutes reliées entre elles selon le schéma suivant.



Si l'on veut pouvoir transférer les données d'une case mémoire dans celle de droite ou de gauche, il faut interconnecter les bascules entre elles : on obtient alors un **registre à décalage**.

Par exemple, on voit dans le schéma ci-dessus que, si on relie Q_A à D_B, Q_B à D_C et Q_C à D_D, on obtient un registre à décalage à droite, c'est à dire qu'à chaque impulsion de l'horloge, le bit contenu dans la bascule A sera transféré dans B, celui contenu dans B sera transféré dans C et ainsi de suite.

Si l'on veut obtenir un registre à décalage à gauche, il faut relier Q_D à D_C, Q_C à D_B et Q_B à D_A.

A partir de ce principe, on peut compléter le schéma pour obtenir un registre permettant soit le décalage à droite, soit le décalage à gauche, soit le chargement parallèle (4 valeurs binaires a, b, c et d sont mémorisées dans les 4 bascules, sur la même impulsion d'horloge), soit la mémorisation (les données mémorisées ne changent pas au moment des impulsions d'horloge). Un tel registre est dit "registre universel".

Pour obtenir ces 4 modes différents de fonctionnement, il faudra bien sur disposer de 2 entrées de commande pour choisir l'un d'entre eux.

II. Applications

- Conversion série/parallèle : si l'on envoie les 4 bits d'un nombre les uns après les autres (c'est à dire en série) sur l'entrée D_A, au bout de 4 impulsions d'horloge, on les retrouvera présents en même temps (c'est à dire en parallèle) sur les 4 sorties des bascules.

- Multiplication (ou division) par 2 : pour multiplier (ou diviser) un nombre par 2 il faut envoyer un "0" sur D_A (ou sur D_D)et décaler tous les autres bits vers la droite (ou la gauche).

- Ligne à retard numérique : si l'on veut retarder une donnée de n tops d'horloge, il suffit de l'envoyer sur D_A et de la lire sur la sortie de la $n^{ième}$ bascule, en mode décalage à droite.

NUMERATION : CHANGEMENTS DE BASE

1) Donner le rang et le poids de chaque chiffre et convertir ces nombres en décimal

   $(642)_7$; $(AB7)_{12}$; $(10111001)_2$; $(2102)_3$; $(275)_8$; $(BAC)_{16}$; $(10DA)_{16}$

2) Convertir les nombres ci-dessous (donnés en base 10) dans la base indiquée

   194 en base 2 ; 254 en base 2 ; 1025 en base 16 ; 4218 en base 16 ; 1234 en base 8

3) Conversion rapide vers une base 2^n

   $(B7AE)_{16} \rightarrow 2$; $(DCBF)_{16} \rightarrow 2$; $(56349)_{16} \rightarrow 2$; $(5567)_8 \rightarrow 2$; $(ABCD)_{16} \rightarrow 8$;

   $(2074)_8 \rightarrow 16$; $(1110100000111010)_2 \rightarrow 16$; $(1110100000111010)_2 \rightarrow 8$

4) Déterminer la base b pour que les égalités ci-dessous soient vraies.

   $(132)_b = (30)_{10}$; $(A04)_b = (1444)_{10}$; $(2A)_{16} = (36)_b$

5) Déterminer les plus petites bases possibles (a et b) pour que les égalités ci-dessous soient vraies.

   $(101)_a = (401)_b$; $(101)_a = (10001)_b$; $(12)_a = (1002)_b$

6) A quelle condition un nombre écrit en base quelconque est-il pair en base 10 ?
   (discuter selon la base)

7) Convertir les nombres suivants en décimal

   $(1101,011)_2$; $(765,42)_8$; $(FAC,9D)_{16}$

8) Convertir les nombres ci-dessous dans la base indiquée

   $(14,65)_{10} \rightarrow 8$ (3 chiffres après la virgule) ; $(42,42)_{10} \rightarrow 2$ (7 chiffres après la virgule)

   $(69,23)10 \rightarrow 16$ (3 chiffres après la virgule)

   Pourquoi faut-il garder 7 chiffres après la virgule pour convertir $(42,42)_{10}$ en binaire ?

   $(11011000110,001011011)_2 \rightarrow 16$; $(1011110100,1110111)_2 \rightarrow 8$

OPERATIONS EN DIFFERENTES BASES

1) Effectuer les additions suivantes

   en binaire
   10101010 + 11001110 ; 110111 + 101110 + 110011 ; 1110011 + 1111111 + 1101011 + 101110
   en octal (base 8) : 465 + 673 ; 276 + 653 + 25
   en hexadécimal (base 16) : B796 + CAFE ; 8965 + 3979 ; 324 + 697 + B2A

2) Effectuer les soustractions suivantes en binaire

   11101101010 - 110111100 ; 10110001 - 10011111 ; 1101111 - 1110100

3) Effectuer les multiplications suivantes en binaire

   1101101 * 11001 ; 11010110 * 101001

4) Effectuer les divisions suivantes en binaire

   101100 / 101 (5 chiffres après la virgule) ; 101011010 / 1101 (4 chiffres après la virgule)

5) a) Combien peut-on écrire de nombres différents sur 1bit, 2 bits, 3 bits, n bits ?

   b) Une mémoire comporte 13 fils d'adresses (chacun d'entre eux pouvant être à 0 ou 1)
   Combien peut on mémoriser de données différentes dans cette mémoire ?
   Si l'adresse basse est 0, quelle sera l'adresse haute en hexa ?
   Convertir sa capacité mémoire en hexa sans effectuer la division par 16.

   c) Mêmes questions pour une mémoire comportant 15 fils d'adresses.

   d) Un système informatique comporte 4 mémoires (M_1, M_2, M_3 et M_4), les 2 premières ayant 13 fils d'adresses, les 2 dernières ayant 15 fils d'adresses. Elles sont rangées par ordre croissant, l'adresse basse de la première étant 0.
   Donner dans un tableau les adresses basses et hautes de chaque mémoire.
   Quelle est la capacité mémoire totale de ce système ?
   Combien de fils d'adresses sont nécessaires en tout sur le microprocesseur qui gère ce système ?

6) On veut mémoriser une longueur L pouvant varier de 0 à 15 cm. Pour cela on utilise un **capteur** qui la transforme en une **tension** proportionnelle à L puis un **convertisseur analogique numérique** qui la transforme en un nombre binaire codé sur n bits : ce nombre binaire pourra donc être mis en mémoire.

   a) Combien de bits seront nécessaires pour mesurer cette longueur avec une précision au moins égale à 0, 1 mm ?
   b) Quelle est la précision exacte obtenue ? (en mm)
   c) Quelle est la relation entre L et le nombre mémorisé x noté en décimal ?
   d) Par quel nombre est codée une longueur de 10 cm ? (en décimal, hexa et binaire) (arrondir à la valeur entière la plus proche)
   e) Quelle longueur est codée par le nombre $(72C)_{16}$?

NOMBRES SIGNES ET CODES

1) a) Coder les nombres suivants en binaire signé (sur 1 octet y compris le bit de signe)
-1 ; -29 ; -40 ; -127 ; -128

b) Si on écrit les nombres signés en hexa, à quelles plages de valeurs correspondent les nombres positifs et les nombres négatifs ?

2) Opérations en nombres signés (sur 1 octet y compris le bit de signe)

a) Si l'on effectue une addition entre 2 nombres signés et que le résultat doit lui aussi tenir sur 8 bits, quels sont les résultats faux (qui dépassent les valeurs limites) ?
Par quelle comparaison entre les bits de signe des 2 nombres et celui du résultat peut-on détecter cette erreur (dite "erreur de débordement" **à ne pas confondre** avec le bit de débordement qui serait le $9^{ème}$ bit) ?

Rem : comme une soustraction se ramène à une addition en utilisant la méthode du complément à 2, les résultats seront identiques pour les 2 opérations.

b) Effectuer les soustractions suivantes par la méthode du complément à 2, tous les nombres étant codés en binaire signé.
Le résultat doit être lui aussi en binaire signé sur 1 octet. S'il y a une erreur de débordement, le signaler, sinon convertir le résultat en décimal.

00110100 - 11001101 ; 00110100 - 10000011 ; 11101110 - 00110011
10000001 - 00001000 ; 10001010 - 10000110 ;

3) Le code Gray

a) Ecrire en code Gray les nombres de 0 à 15 et en déduire le code Gray de 17, 24, 31.

b) Méthode de codage binaire vers code Gray

On veut coder $X_b = b_n\ b_{n-1}\ b_{n-2} \ldots b_{n-i}\ b_{n-i-1} \ldots b_1\ b_0$ en code Gray :
soit $X_g = g_n\ g_{n-1}\ g_{n-2} \ldots g_{n-i}\ g_{n-i-1} \ldots g_1\ g_0$.

Appliquer la règle suivante : pour i variant de 0 à n - 1, si b_{n-i} et b_{n-i-1} ont la même valeur alors $g_{n-i-1} = 0$, sinon $g_{n-i-1} = 1$. Ex : 11001011_B donne 10101110_G

Coder en code Gray les nombres suivants : 45 ; 100 ; 120.

c) Méthode de codage code Gray vers binaire

On compte le nombre de 1 à partir du poids fort jusqu'au bit dont on veut la valeur binaire : si ce nombre est pair on note 0 sinon 1. Ex : 1011en Gray donne 1101 en binaire.

Donner la valeur décimale des nombres suivants codés en code Gray.
11001100 ; 10101010 ; 10000001 ;

4) Coder en BCD les nombres suivants : 97 ; 724 ; 2009 ; 6308

LES PORTES LOGIQUES EN ELECTRONIQUE

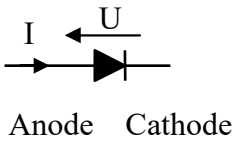
Les portes logiques actuelles ainsi que tous les circuits intégrés sont réalisés à partir de transistors gravés sur une puce de silicium et la technologie en est assez complexe mais on peut en donner un équivalent électrique simple à partir de diodes et de transistors.

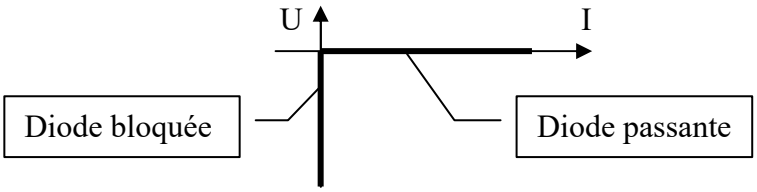
1) portes à base de diodes

Pour comprendre comment on réalise des portes à base de diodes, il faut comprendre comment fonctionne … une diode en commutation !

Une diode en commutation est équivalente à un **interrupteur ouvert ou fermé selon le sens du courant** : si le courant peut passer dans le sens de la flèche, c'est à dire de l'anode vers la cathode (de gauche à droite sur le schéma ci-dessous), alors l'interrupteur est fermé : $I > 0$ et $U = 0$ (la diode est dite passante). Sinon l'interrupteur est ouvert : $I = 0$ et $U < 0$ (la diode est dite bloquée).

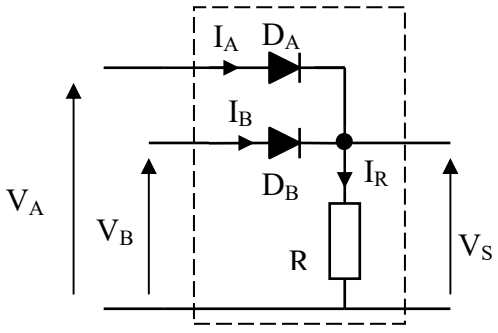
On obtient donc la courbe U = f(I) ci-dessous.

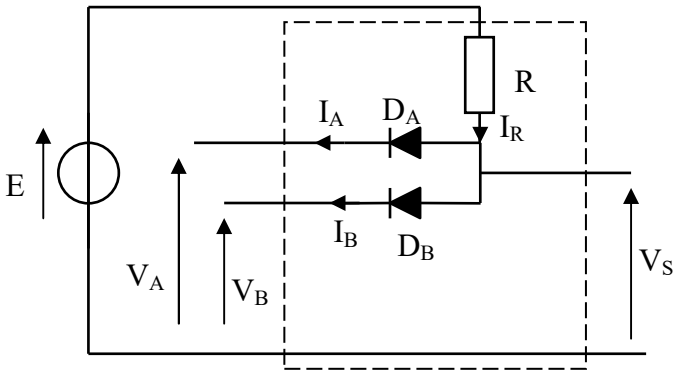




En fait, pour que la diode soit passante (interrupteur fermé) il faut que U soit au moins égale à 0,6 V pour du silicium (on appelle cette tension la tension de seuil), mais on négligera souvent cette petite tension.

A partir de ces explications, compléter les tableaux correspondant aux 2 schémas ci-dessous en indiquant l'état (P)assant ou (B)loqué des diodes, les valeurs des courants et de V_S selon les valeurs de V_A ou V_B (E = 5 V et R = 5 kΩ). En déduire le type des portes ainsi réalisées.





V_B	V_A
0	0
0	5
5	0
5	5
type de porte	

état de D_B	état de D_A	I_B	I_A	I_R	V_S

état de D_B	état de D_A	I_B	I_A	I_R	V_S

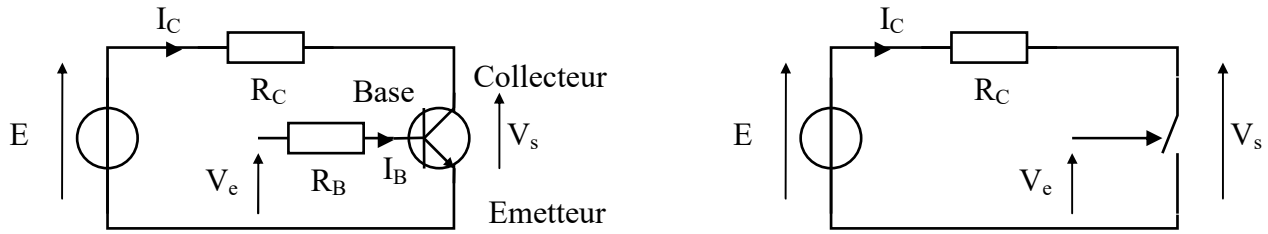
2) Portes à base de transistors

Un transistor en commutation peut être considéré comme un interrupteur **entre le collecteur et l'émetteur,** le bouton poussoir étant le courant dans la base (voir le schéma ci-dessous).

- Si l'on envoie du courant dans la base en lui appliquant une tension positive Ve, l'interrupteur se ferme : on dit que le transistor est saturé. On obtient :$V_S = 0$ et $I_C = E/R_C$.

- Si l'on n'envoie pas de courant dans la base, l'interrupteur s'ouvre : on dit que le transistor est bloqué. On obtient : $I_C = 0$ et $V_S = E$.

Les résistances R_B et R_C permettent de régler l'intensité des courants dans la base et dans le collecteur.



Compléter le tableau ci-dessous et en déduire le type de porte ainsi réalisée.

V_e	état du transistor	V_S
0		
5		

3) A partir des 2 exercices précédents, dessiner le schéma d'une porte NOR puis d'une porte NAND

FONCTIONS LOGIQUES ET SIMPLIFICATION -1

1) Soit 4 variables logiques, A, B, C, D. Ecrire de 2 manières différentes (la 1ère avec seulement des ET ou des OU, la 2ème avec l'autre opérateur logique et des NON) des **égalités** qui sont vérifiées **si et seulement si** :

   a) Les 4 variables sont à 1
   b) Les 4 variables sont à 0
   c) Au moins l'une d'entre elles est à 1
   d) Au moins l'une d'entre elles est à 0

2) Simplifier les expressions suivantes :

$S_1 = (A + B).(\overline{A} + \overline{B})$

$S_2 = A.B + \overline{A}.\overline{B} + \overline{A}.B$

$S_3 = (A + \overline{B}).(A + B) + C.(\overline{A} + B)$

$S_4 = (A + C + D).(B + C + D)$

$S_5 = (A.\overline{B} + A.B + A.C)(\overline{A}.\overline{B} + A.B + A.\overline{C})$

$S_6 = (A + \overline{B} + C).(A + \overline{C}).(\overline{A} + \overline{B})$

3) Calculer le complément de S_1, S_5, S_6 et le simplifier

4) Donner l'équation des fonctions NON, ET, OU en n'utilisant que des portes NAND puis en n'utilisant que des portes NOR

5) Donner les équations des fonctions S_1, S_5 et S_6 en n'utilisant que des portes NAND à 2 entrées puis en n'utilisant que des portes NOR à 2 entrées. Préciser le nombre de portes nécessaires dans chaque cas et en déduire la meilleure solution.

6) Simplifier les expressions suivantes :

$S_1 = A.B.C + A.\overline{B}.C + A.B.\overline{C}.D$

$S_2 = A + B.C + \overline{A}.(\overline{B} + \overline{C}).(A.D + C)$

$S_3 = (A + B + C).(A + B + \overline{C}).(\overline{A} + B + C).(\overline{A} + B + \overline{C})$

7) Démontrer les égalités suivantes

$\overline{A.C + B.\overline{C}} = \overline{A}.C + \overline{B}.\overline{C}$

$(A + B).(\overline{A} + C).(B + C) = (A + B).(\overline{A} + C)$

$\overline{(A + C).(B + \overline{C})} = (\overline{A} + C).(\overline{B} + \overline{C})$

Rem : On peut toujours rajouter "0" (soit $X.\overline{X}$) à une expression ou la multiplier par "1" (soit $X + \overline{X}$) sans la modifier. Cela peut être utile pour faire apparaître des termes communs (qui sinon n'apparaîtraient pas) et permettre ainsi des mises en facteurs ou des simplifications.

FONCTIONS LOGIQUES ET SIMPLIFICATION – 2

1) Réaliser la fonction OU exclusif ($X \oplus Y = X.\overline{Y} + \overline{X}.Y$) avec 5 NAND puis avec 4 NAND.
   (Pour le faire avec 4 NAND, rajouter des quantités nulles et factoriser… mais pas trop !)

2) Soient 3 variables A, B, C :
   Réaliser une fonction qui donne 1 si le nombre de variables à 1 est impair
   (simplifier avec des OU exclusifs)

Une façon simple d'énoncer un problème sur n variables est de leur affecter un rang, c'est à dire de les considérer comme formant un nombre.
Dans la suite des exercices, la variable A aura toujours le rang 0, la variable B le rang 1, etc…
Si on utilise 3 variables, le nombre N s'écrira en binaire $N = (CBA)_2$ avec $0 \leq N \leq 7$.
Au lieu de décrire une fonction S par tous les cas de figure possibles, il suffira de dire que S doit être égale à 1 pour certaines valeurs de N.
Par exemple au lieu de demander de réaliser une fonction S qui donne 1 ssi 2 variables sur 3 sont à 1, il suffira de dire : on veut que S = 1 ssi N = 3 ou 5 ou 6.

3) Réaliser une fonction S=f(C, B, A) telle que S = 1 si : a) $N \geq 5$; b) $N \leq 2$; c) $2 < N \leq 6$

4) Complémenteur à 2 sur 3 bits

   On veut réaliser un circuit qui donne le complément à 2 d'un nombre codé sur 3 bits (CBA)
   a) Ecrire la table de vérité de chacune des 3 sorties C' B' A'.
   b) Simplifier les équations de ces 3 sorties.

5) Mêmes questions pour réaliser un circuit qui code en code Gray un nombre binaire sur 3 bits.

6) Soit N un nombre binaire codé sur 4 bits (DCBA)

   Remplir directement les tableaux de Karnaugh et donner les équations simplifiées de S dans les cas suivants :
   a) S = 1 ssi $N \geq 10$
   b) S = 1 ssi N = 0, 4, 8, 10, 12 ou 14
   c) S = 1 ssi N = 0, 2, 8 ou 10

7) Soit N un nombre binaire codé sur 5 bits (EDCBA)
   Remplir directement les tableaux de Karnaugh et donner les équations simplifiées de S

   a) S = 1 ssi N = 0, 1, 9, 11, 13, 15, 16, 17, 20, 21, 25, 26, 27, 30, 31
   b) S = 1 ssi N = 0, 2, 8, 10, 13, 15, 16, 18, 24, 25, 26, 29, 31 avec 5, 7, 9, 12, 28 indifférents

OPERATIONS ARITHMETIQUES

1) Soit $N = (DCBA)_2$ avec $N < 10$

   On veut effectuer une multiplication par 2 et donc obtenir $N' = 2N$ avec N' codé directement en BCD (c'est à dire sur 8 bits : H' G' … A')

   a) Remplir les tables de vérité des sorties de ce circuit.
   b) Remplir les tableaux de Karnaugh nécessaires et en déduire les équations simplifiées des sorties.

2) Additionneur complet

   Pour faire une addition en binaire (et d'ailleurs quelle que soit la base), on additionne les 2 chiffres d'une même colonne en tenant compte d'une éventuelle retenue qui peut venir de la colonne de droite : on obtient donc le chiffre du résultat et aussi une éventuelle retenue qu'il faudra ajouter aux chiffres de la colonne de gauche.

   On veut réaliser un circuit qui additionne 2 bits (A_i et B_i) en tenant compte d'une éventuelle retenue R_{i-1}. Ce circuit doit donc générer la somme S_i et l'éventuelle retenue R_i à transmettre à la colonne de gauche.

   a) Remplir la table de vérité de S_i et R_i
   b) Remplir les tableaux de Karnaugh et en déduire les équations simplifiées de S_i et R_i.
   c) Dessiner le schéma de ces 2 fonctions réunies en un seul bloc fonctionnel : l'additionneur complet.
   d) Dessiner le schéma d'un additionneur de 2 nombres de 4 bits en utilisant 4 blocs fonctionnels identiques.

3) Soustracteur complet

   On veut réaliser un circuit qui effectue la soustraction $A_i - B_i$ en tenant compte d'une éventuelle retenue R_{i-1}. Ce circuit doit donc générer la différence D_i et l'éventuelle retenue R_i à transmettre à la colonne de gauche.

   Mêmes questions que ci-dessus.

4) Additionneur Soustracteur

   a) Réaliser un circuit qui inverse ou non l'état d'une entrée E selon qu'un bit de commande C est à 1 ou à 0 : si C = 0 on veut S = E, si C = 1 on veut $S = \overline{E}$.

   b) En utilisant cette fonction et un additionneur sur 4 bits (vu dans l'exercice 2 question d), réaliser un circuit qui effectue l'addition de 2 nombres de 4 bits (A + B) si un bit de commande C est à 0 et la soustraction (A – B) si C = 1.

COMPARAISON ET AFFICHAGE

1) Comparateur de bits

On veut construire un circuit qui compare 2 bits

Il devra comporter : 1 entrée de validation : G
2 entrées : A et B (bits à comparer)
3 sorties : S(upérieur), E(galité) et I(nférieur)

Si G = 0 $\Rightarrow$ S = E = I = 0 $\forall$ A et B (circuit bloqué)
Si G = 1 $\Rightarrow$ A > B $\Rightarrow$ S = 1 ; E = I = 0
A = B $\Rightarrow$ E = 1 ; S = I = 0
A < B $\Rightarrow$ I = 1 ; S = E = 0
Calculer les équations simplifiées de S, E et I et dessiner le schéma correspondant.

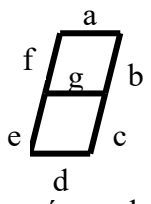
2) Comparateur de nombres

On veut comparer 2 nombres de 4 bits (non signés) : A (A_3, A_2, A_1, A_0) et B (B_3, B_2, B_1, B_0) en respectant les mêmes consignes que ci-dessus.
Proposer un schéma en utilisant les cellules de bases vues dans le 1er exercice et les portes nécessaires.

3) Afficheur 7 segments

A partir des sorties S, E et I du comparateur précédent, on veut afficher sur un afficheur 7 segments les lettres suivantes :

si G = 0 si A > B si A = B si A < B



repérage des LEDs de l'afficheur.

(un "1" sur le segment allume la LED correspondante)

a) Donner la table de vérité des 7 segments en fonction de S, E et I
(S, E,I et a, b, c, d, e, f et g de la gauche vers la droite)
b) Remplir les tableaux de Karnaugh nécessaires et donner les équations simplifiées des 7 sorties. (2 lignes, 4 colonnes et S, E, I de gauche à droite)
c) Dessiner le schéma en n'utilisant que des opérateurs NOR à 2 entrées.

4) Décodeur 7 segments

On veut afficher un chiffre décimal (codé sur 4 bits : DCBA) sur un afficheur 7 segments.
Déterminer les équations simplifiées de chaque segment en fonction de DCBA.

MULTIPLEXAGE, DECODAGE ET DEMULTIPLEXAGE

1) Multiplexeur

un multiplexeur est une sorte d'aiguillage, c'est à dire un circuit qui permet de sélectionner une entrée parmi n grâce à son adresse, et de faire apparaître en sortie l'état de cette entrée.

ex : avec 2 entrées d'adresses (B, A), on peut sélectionner une entrée E_i parmi 4 (E_0 …E_3) :
si BA = $10_2 = 2_{10}$, on veut que S = E_2

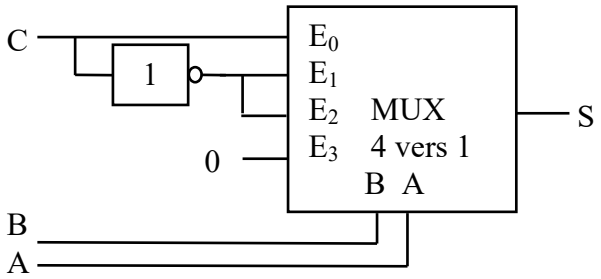
De façon générale, avec n entrées d'adresses, on peut sélectionner une entrée parmi 2^n.
On aura donc un multiplexeur 2^n vers 1.

Ecrire la table de vérité de la sortie S d'un multiplexeur 4 vers 1 en fonction des entrées d'adresses B et A et des entrées de données E_0 à E_3, et en déduire l'équation de S.
Dessiner le schéma correspondant.

2) Utilisation d'un multiplexeur
On veut réaliser la fonction suivante : S = 1 ssi N = 0 ou 3 ou 5 ou 7, N étant un nombre codé sur 3 bits (C, B, A).
a) Ecrire la table de vérité de S et en déduire le montage utilisant le moins possible de portes logiques.
b) Réaliser la même fonction avec le multiplexeur adéquat.
c) De façon générale, combien existe-t-il de <u>fonctions</u> différentes de n variables ?
En déduire l'intérêt d'un multiplexeur.

3) Soit le circuit suivant

Quelle fonction réalise-t-il ?



4) Réaliser la même fonction S que dans l'exercice 2) a) avec un multiplexeur 4 vers 1 et un simple inverseur en plus.

5) Décodeur/Démultiplexeur

Un décodeur est un circuit qui dispose de n entrées et de 2^n sorties et qui fonctionne de la façon suivante :
Toutes les sorties sont dans l'état inactif sauf celle dont on donne l'adresse sur les entrées d'adresses.

exemple sur un décodeur 3 vers 8 :
les entrées d'adresses sont notées C, B, A (du poids fort au poids faible)
les sorties sont notées Y_0 à Y_7.
Si CBA = $101_2 = 5_{10}$, Y_5 est activée, les autres sorties restant inactives.

Si on rajoute une entrée de validation G ce circuit est appelé démultiplexeur (le circuit ne fonctionne comme indiqué ci-dessus que si cette entrée est activée, sinon toutes les sorties restent dans l'état inactif quel que soit l'état des entrées d'adresses)

Rem : en fait dans les docs techniques, ces circuits sont confondus et on utilise le même composant en connectant ou non l'entrée de validation selon les besoins.

Le circuit 74LS138 est un décodeur/démultiplexeur 3 vers 8 qui dispose de 3 entrées de validation G_1, G_{2A}, G_{2B}, de 3 entrées d'adresses C, B, A, de 8 sorties Y_0 à Y_7 et qui fonctionne comme suit :

Le circuit n'est validé que si $G_1 = 1$ et $G_{2A} = G_{2B} = 0$, sinon : $Y_i = 1$ quels que soient A, B et C (sorties inactives : circuit bloqué)
Si le circuit est validé, $Y_i = 0$ si CBA = i_{10}, les autres sorties restant à 1
(l'état actif est 0 sur les sorties)

a) Donner les équations et le câblage des sorties Y_i en fonction de G_1, G_{2A}, G_{2B}, C, B et A.

b) Réaliser la même fonction S que dans l'exercice 2) a) avec un 74LS138 et la ou les porte(s) nécessaire(s) à 2 entrées.

c) Comparer les 3 solutions permettant de réaliser la même fonction.

DECODAGE D'ADRESSES (SIMPLIFIE) D'UN SYSTEME INFORMATIQUE

On suppose qu'on utilise un microprocesseur comprenant 20 fils d'adresses.
Donner l'espace mémoire adressable ainsi que les adresses basses et hautes (en hexa).

Ce microprocesseur doit gérer les périphériques suivants (classés par ordre croissant dans l'espace mémoire à partir de l'adresse 00000h):

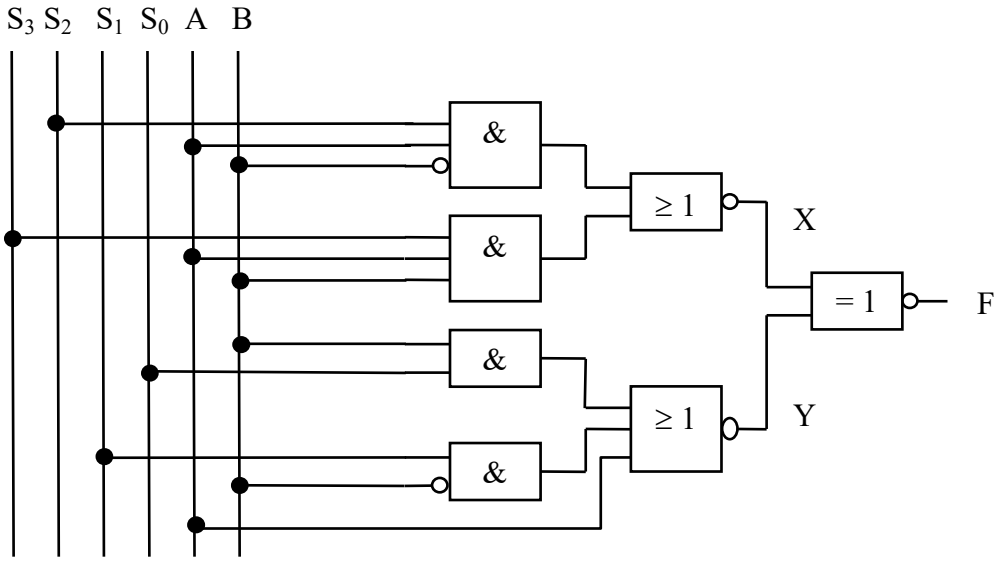
- 2 EPROM de 8 K@
- 4 RAM de 8K@
- 1 DUART (périphérique de liaison série) (on admettra qu'il occupe 8 K@ pour l'adressage)
- 1 port d'entrées-sorties (I/O) (on admettra qu'il occupe 8 K@ pour l'adressage)

Chaque composant est validé par le passage à l'état bas d'une broche appelée CS (Chip Select).

a) Exprimer en hexa la taille mémoire des composants et noter les numéros des fils d'adresses communs à tous les composants.
b) Noter dans le tableau ci-dessous l'adresse basse et l'adresse haute de chaque composant.
c) compléter le tableau en indiquant précisément l'état des fils d'adresses dans chaque ligne.
d) Quels sont les fils d'adresses qui différencient les différents composants ?
e) En déduire le schéma d'adressage du système informatique basé sur un 74LS138.
f) Quel est l'espace mémoire restant et à quels fils correspond-il ?
g) quelle est l'adresse de la donnée lue par le microprocesseur si on programme l'adresse 80000h ? Conséquence ?
h) Compléter le schéma ci-dessus en reliant correctement ces fils au circuit de décodage afin d'obtenir un adressage non ambigu.
i) On suppose de plus que le microprocesseur possède une broche AS (Address Strobe) qui passe à l'état bas quand l'adresse est bien positionnée sur le bus d'adresses. Compléter le schéma en la reliant au circuit de décodage pour que les différents composants ne soient validés que quand l'adresse est stable sur le bus d'adresses.

			0 à F	0 à F				000 à FFF
		adresses basses et hautes	A_{19} à A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11} à A_0
EPROM 1	@ basse							
	@ haute							
EPROM 2	@ basse							
	@ haute							
RAM 1	@ basse							
	@ haute							
RAM 2	@ basse							
	@ haute							
RAM 3	@ basse							
	@ haute							
RAM 4	@ basse							
	@ haute							
DUART	@ basse							
	@ haute							
I/0	@ basse							
	@ haute							

UNITE LOGIQUE



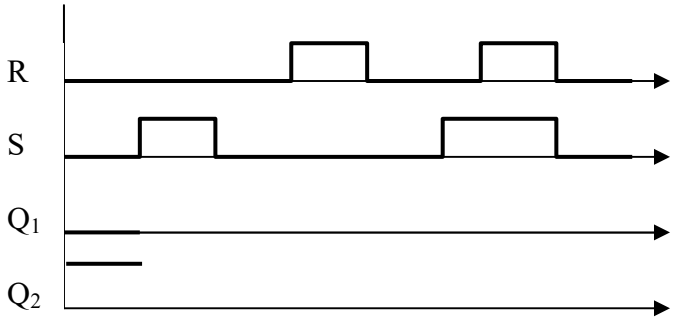
On veut obtenir différentes fonctions F (A, B) selon les valeurs de S_0 à S_3

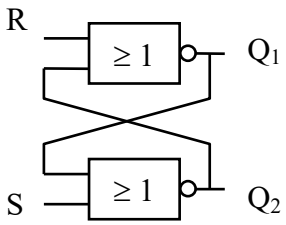
1) Donner l'expression de X en fonction de A, B, et des entrées S_i nécessaires.
2) Donner sous forme de table de vérité l'expression X (A, B) en fonction des entrées S_i ci-dessus. Les entrées S_i nécessaires seront notées par ordre décroissant de la gauche vers la droite.
3) Mêmes questions pour Y.
4) Reporter les résultats ci-dessus dans le tableau ci-dessous et le compléter, sachant que
   $F = \overline{X \oplus Y} = X.Y + \overline{X}.\overline{Y}$
   Les différentes expressions de F (A, B) sont très simples et ne font intervenir qu'une fois maximum les variables A et/ou B.

S_3	S_2	S_1	S_0	X(A, B)	Y(A, B)	F(A, B)

LES BASCULES R S ASYNCHRONES

1) a) Compléter les chronogrammes de la bascule RS ci-contre.



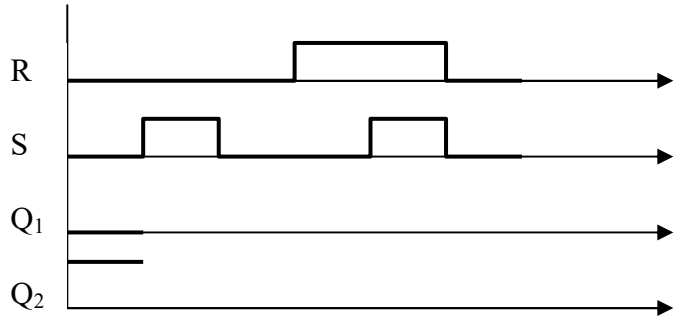


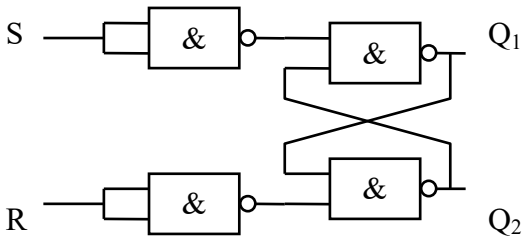
b) Dans quel état sont Q_1 et Q_2 si R = S = 1 ?
Quel est le type de cette bascule ? (marche ou arrêt prioritaire)

c) Que se passe-t-il à la dernière transition ? Indiquer l'état interdit et expliquer pourquoi il est interdit.

---

2) Mêmes questions que ci-dessus.





---

3) Réalisation d'une alarme

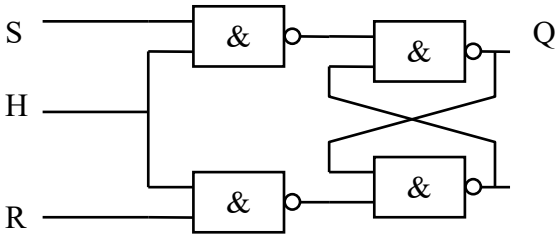
Cahier des charges :
- Elle doit être alimentée par une tension de 5 V.
- Elle doit disposer de 3 capteurs C_1, C_2, C_3 : dès que l'un d'entre eux est activé, l'alarme doit se déclencher.
- Toute alarme doit allumer une lampe L et, de plus, déclencher une sonnerie S si une variable logique notée X est activée à 1.
- Un bouton poussoir (Reset) doit pouvoir éteindre l'alarme sauf si un des 3 capteurs reste activé.

a) Quel type de bascule RS faut-il choisir et pourquoi ?
b) Dessiner le schéma complet de cette alarme en utilisant pour le Reset un bouton poussoir et une résistance. (une entrée logique ne doit pas rester "en l'air" c'est à dire non reliée à un potentiel donné (0 ou 1) mais peut y être reliée à travers une résistance)

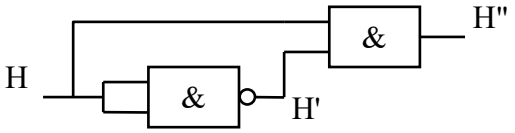
LES BASCULES RS SYNCHRONES

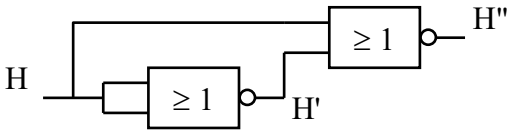
Rem : les sorties Q barre seront notées $\underline{Q}$ pour faciliter l'écriture

1) La bascule ci-contre est une bascule synchronisée sur état :
   - Tant que H = 0, Q conserve l'état précédent (état mémoire).
   - Tant que H = 1, elle se comporte comme une bascule asynchrone.



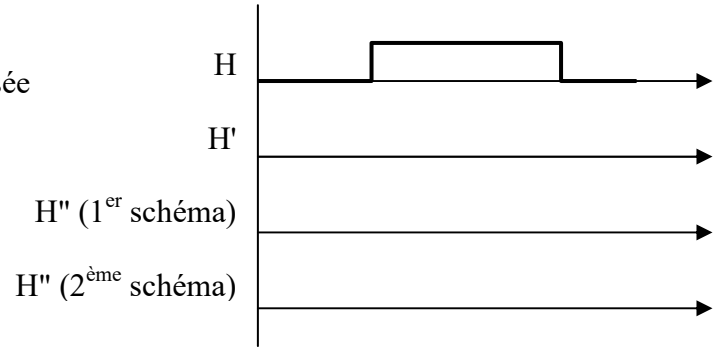
Pour la transformer en une bascule synchronisée sur front montant ou descendant, on insère entre l'entrée H et le point commun aux deux NAND de gauche l'un ou l'autre des montages suivants.





Compléter les chronogrammes ci-dessous en admettant que les portes de gauche (celles dont les sorties sont appelées H') ont un temps de retard double de celles de droite, ce temps restant nettement inférieur à la durée de l'état haut de H.

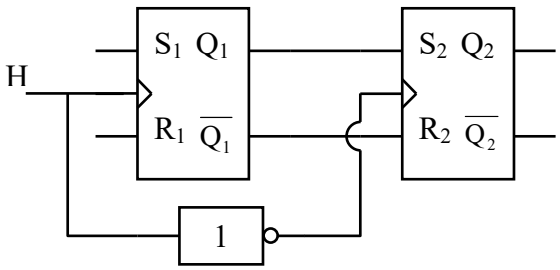
Quel circuit donnera une bascule synchronisée sur front montant / descendant ?

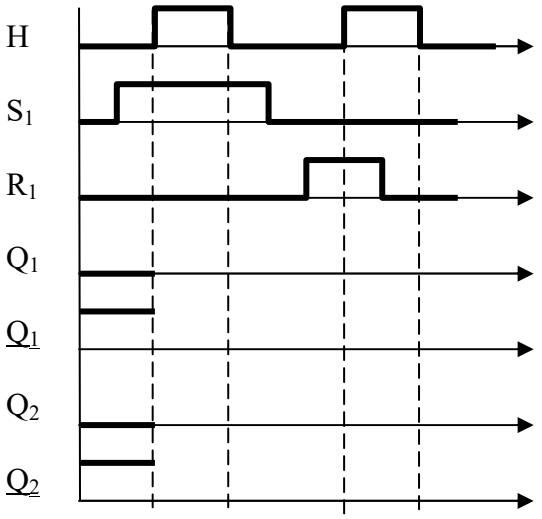


---

2) Bascule RS synchronisée sur impulsion

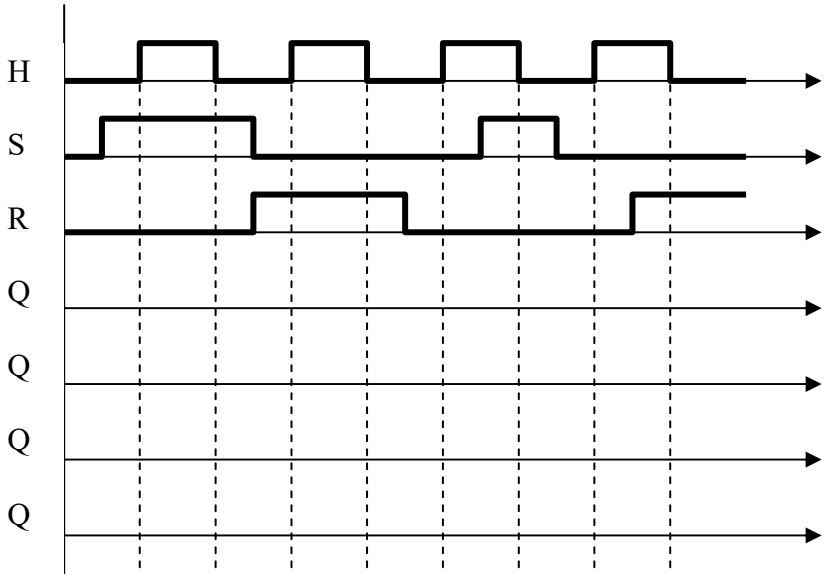
Les 2 bascules sont synchronisées sur front montant. Compléter le chronogramme suivant.





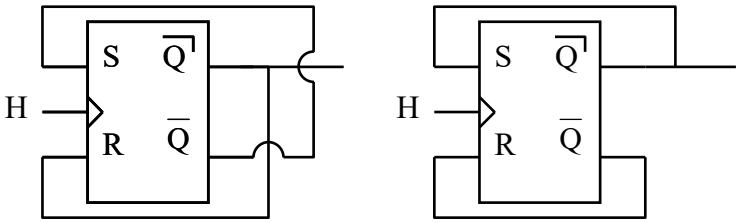
On regroupe ces 2 bascules en un seule bascule RS : expliquer clairement le mode de synchronisation

3) Compléter les chronogrammes suivants selon que la bascule RS est synchronisée sur niveau haut, sur front montant , sur front descendant ou sur impulsion.(Q = 0 à t = 0)



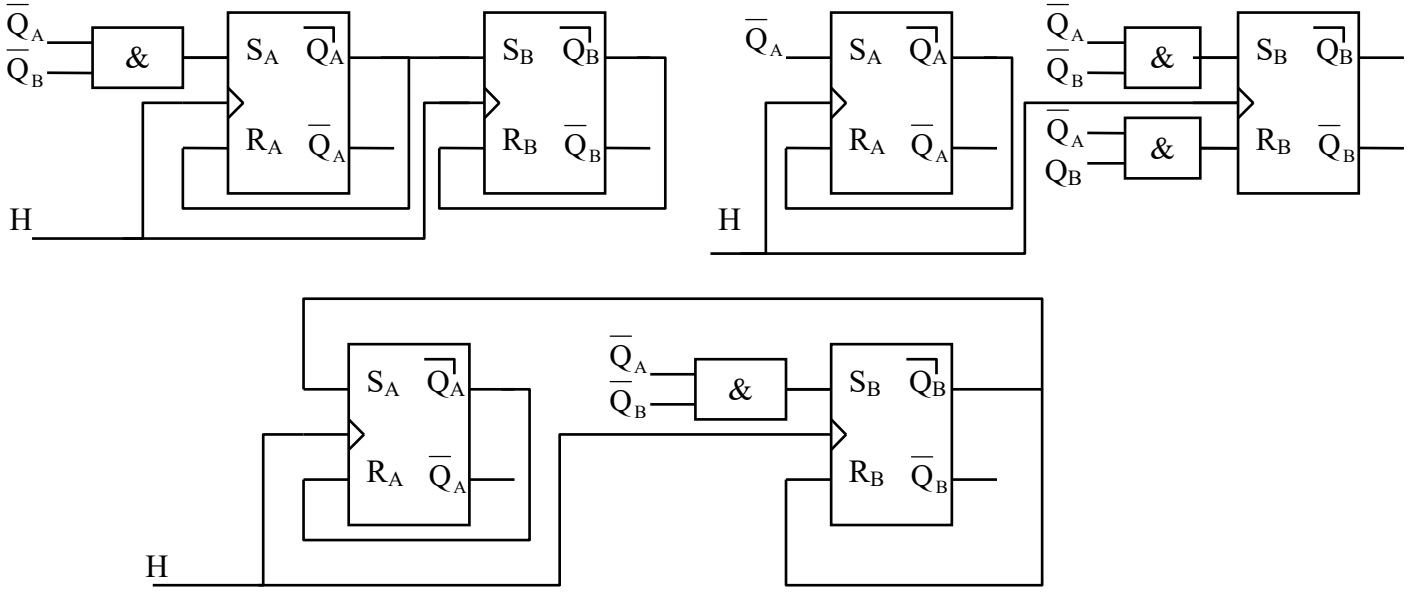
---

4) Les montages suivants utilisent des bascules synchronisées sur impulsion.
Tracer les chronogrammes de la sortie Q en fonction de H (Q = 0 à t = 0)
Dans le circuit de gauche quel est le rapport entre la fréquence du signal présent en Q et celle de H ? Comment appelle-t-on ce montage ?



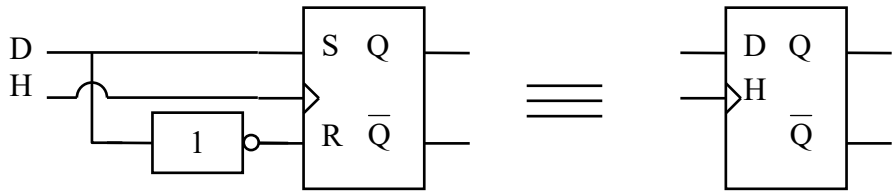
---

5) Les montages suivants utilisent des bascules synchronisées sur impulsion.
Tracer les chronogrammes des sorties Q_A et Q_B en fonction de H ($Q_A = Q_B = 0$ à t = 0)

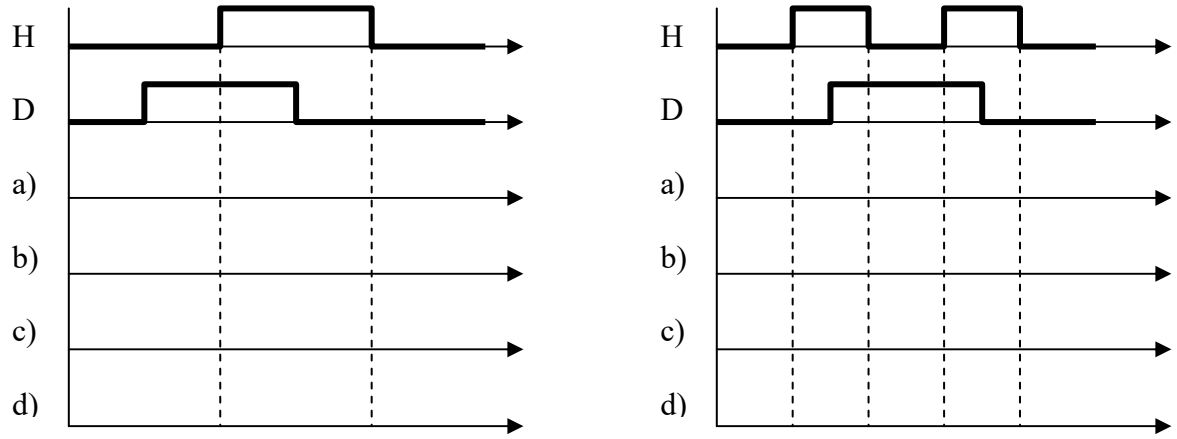


LES BASCULES D

On obtient une bascule D en rajoutant un inverseur entre S et R (voir le schéma)
L'équation d'une bascule D est donc très simple : Q = D : la sortie Q recopie l'état de l'entrée D en fonction du mode de synchronisation. (voir le TD sur les bascules RS pour les différents modes de synchronisation)



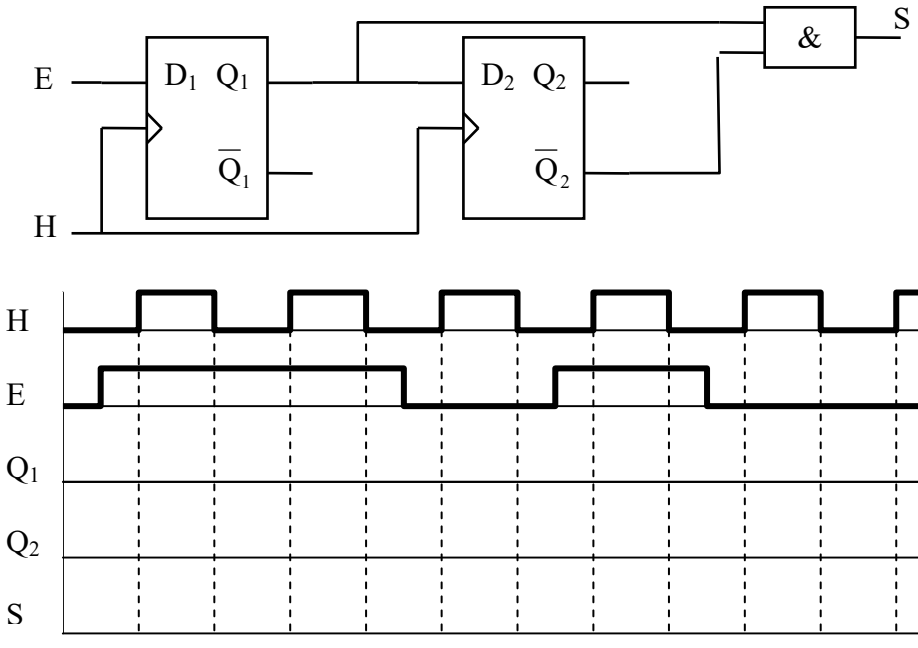
1) Compléter les chronogrammes suivants selon que la bascule D est synchronisée a) sur niveau haut, b) sur front montant , c) sur front descendant, d) sur impulsion.(Q = 0 à t = 0)



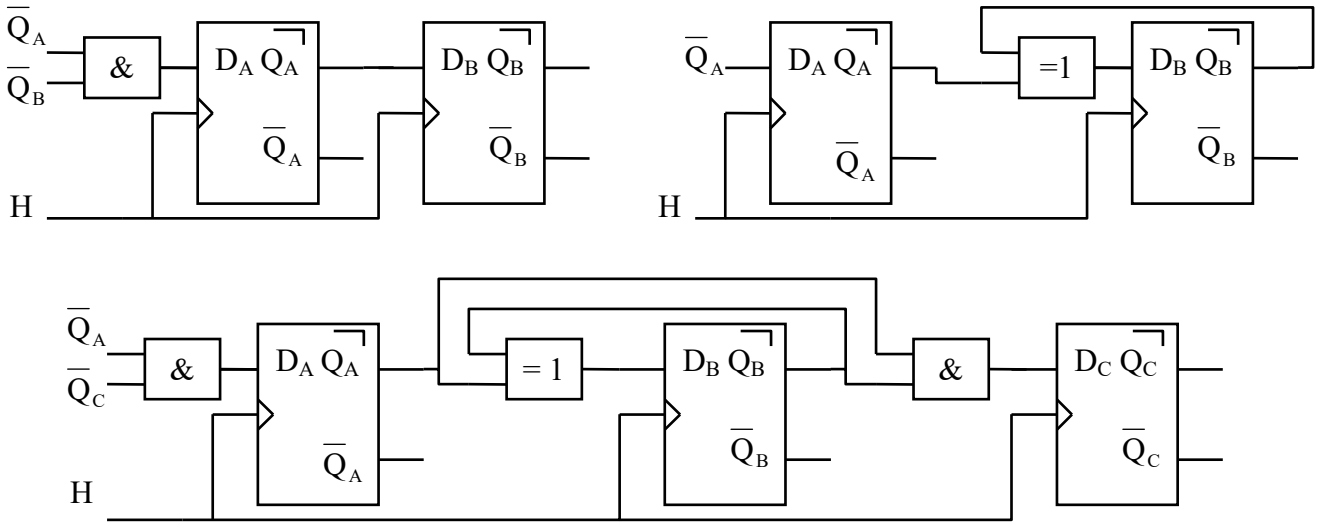
2) Dessiner les chronogrammes de Q en fonction de H (la bascule D est synchronisée sur impulsion et Q = 0 à t = 0)

Dans le circuit de gauche quel est le rapport entre la fréquence du signal présent en Q et celle de H ? Comment appelle-t-on ce montage ?

3) Compléter les chronogrammes de Q_1, Q_2 et S en fonction de H et de E (les bascules D sont synchronisées sur front montant et $Q_1 = Q_2 = 0$ à t = 0)

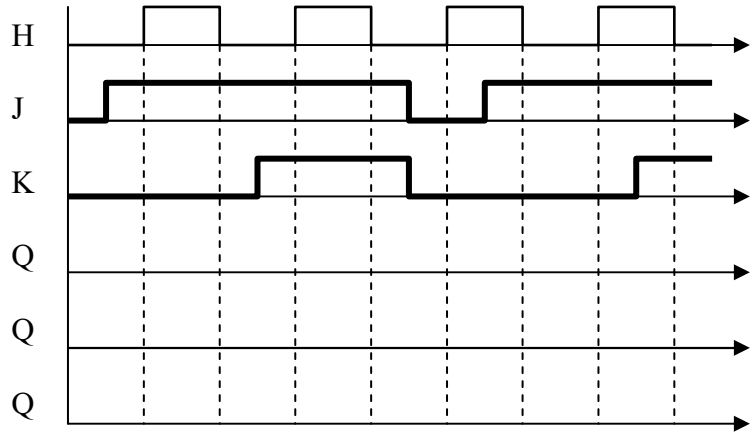


4) Les montages suivants utilisent des bascules synchronisées sur impulsion.
Tracer les chronogrammes des sorties Q_i en fonction de H ($Q_i = 0$ à t = 0)



LES BASCULES JK

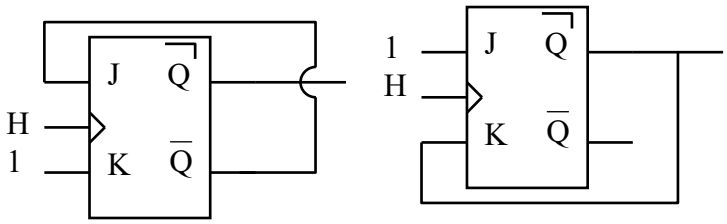
1) Compléter les chronogrammes suivants selon que la bascule JK est synchronisée sur front montant, sur front descendant ou sur impulsion.(Q = 0 à t = 0)



---

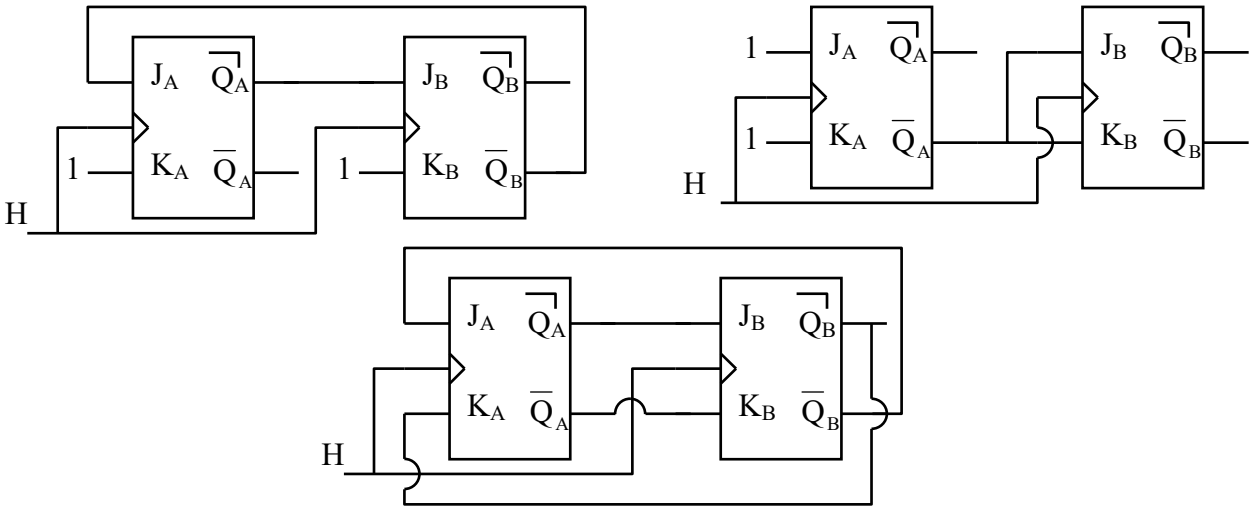
2) Compléter les chronogrammes de Q en fonction de H (bascule JK sur impulsion (Q = 0 à t = 0))

Quel est le rapport entre la fréquence du signal présent en Q et celle de H ? Comment appelle-t-on ce montage ? Trouver une 3ème façon d'obtenir le même rapport.



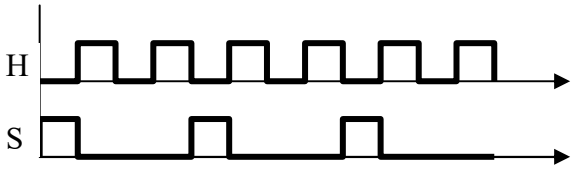
---

3) Les montages suivants utilisent des bascules synchronisées sur impulsion.
Tracer les chronogrammes des sorties Q_A et Q_B en fonction de H ($Q_A = Q_B = 0$ à t = 0)

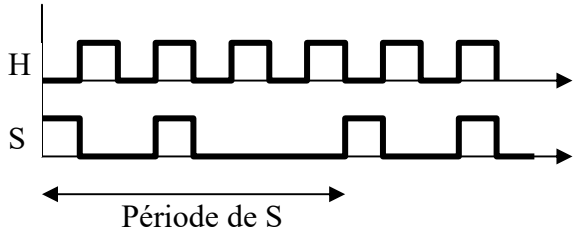


---

4) Réaliser un montage qui délivre le cycle suivant sur sa sortie S en fonction de H (utiliser une bascule JK sur impulsion et 1 porte supplémentaire)

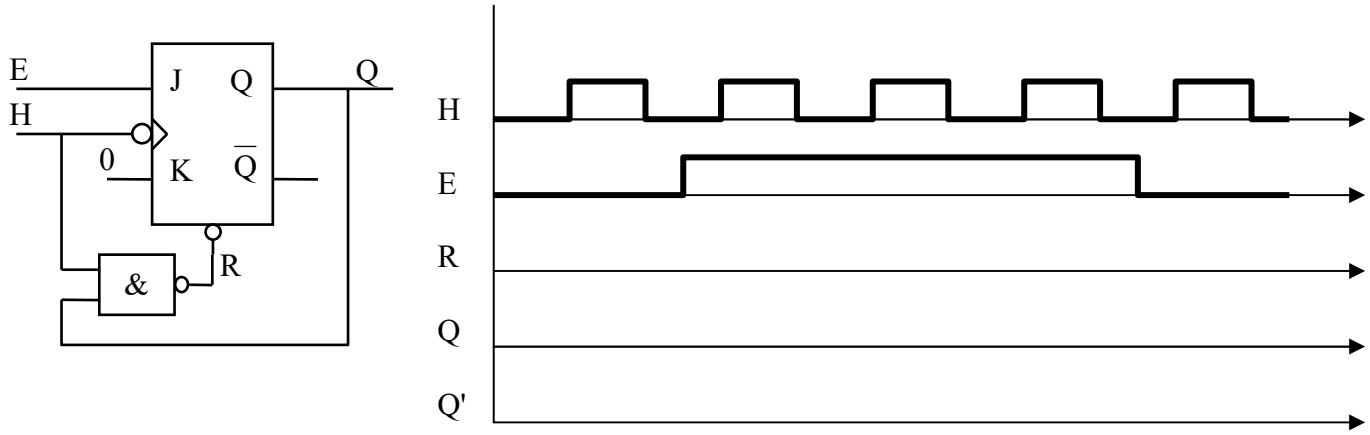


5) Réaliser un montage qui délivre le cycle suivant sur sa sortie S en fonction de H, en utilisant les composants nécessaires.



---

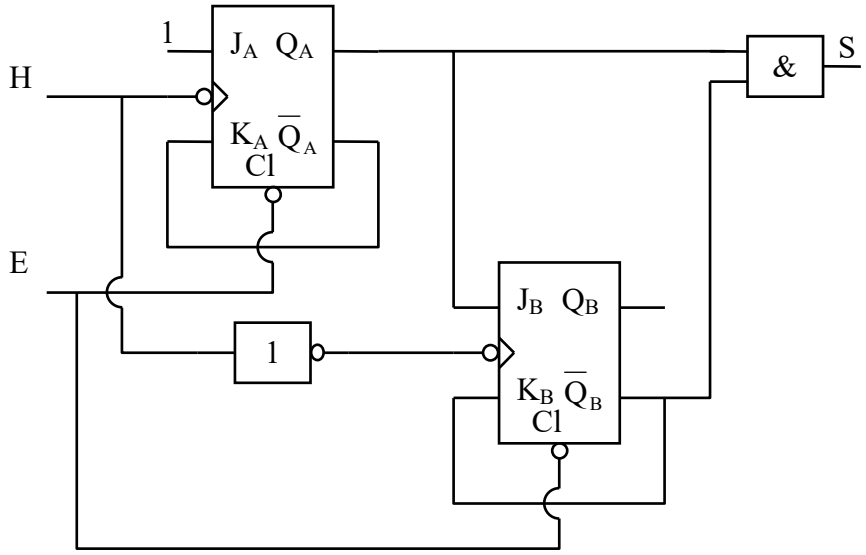
6) Compléter les chronogrammes de R et de Q du circuit suivant :
(la commande Reset est asynchrone)



Tracer sur la dernière ligne le chronogramme de Q' = H.E
Comparer Q et Q' et montrer l'intérêt du montage ci-dessus par rapport à une simple porte ET entre H et E ?

---

7) Compléter les chronogrammes de Q_A, Q_B et S du circuit suivant :
Les chronogrammes de H et de E sont les mêmes que ci-dessus.
(la commande Reset est asynchrone) Quel est l'intérêt d'un tel montage ?



LES COMPTEURS ASYNCHRONES

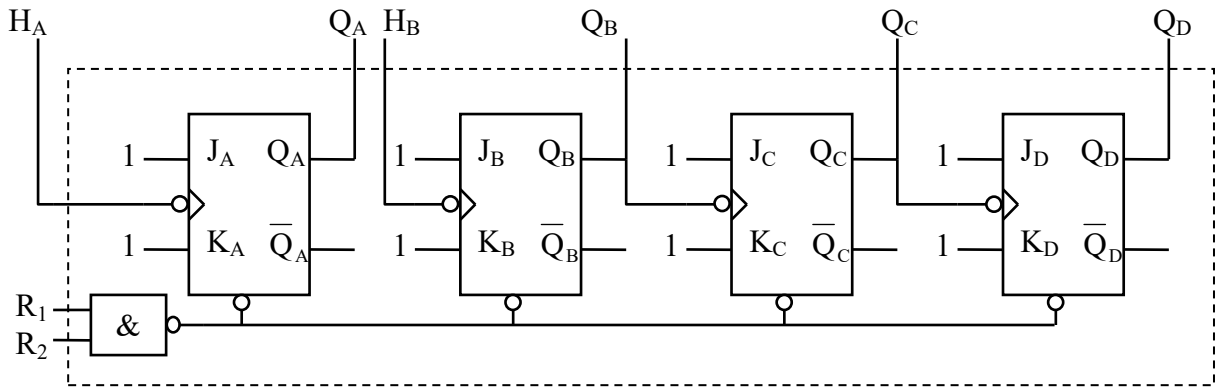
Rappel sur les entrées de forçage des bascules :

En plus des entrées "normales", RS, D ou JK et de l'entrée d'horloge H, les bascules possèdent des entrées de mise à 0 (Reset) ou de mise à 1 (Set ou Preset) en général actives à l'état bas et qui sont prioritaires par rapport à toutes les autres entrées.

Dès que l'entrée Reset = 0, Q = 0 ; Dès que l'entrée Preset = 0, Q = 1
Evidemment, il faut éviter des les mettre toutes les 2 ensemble à 0 !

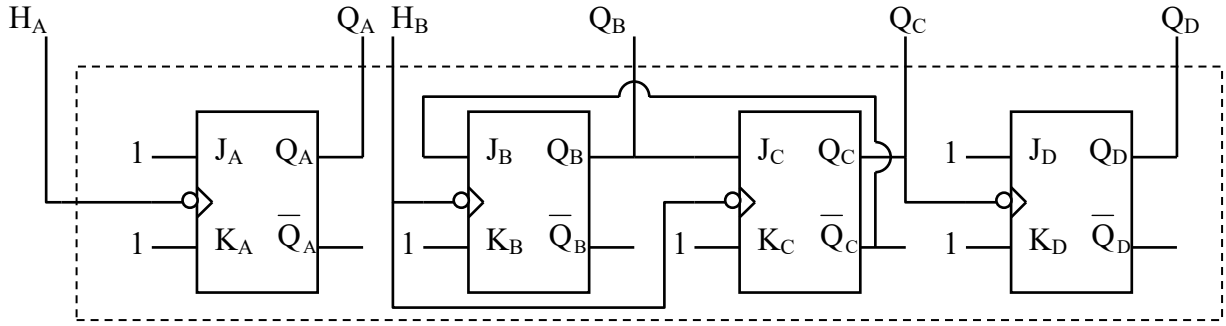
1) Réaliser un compteur asynchrone modulo 9 avec des bascules JK puis avec des bascules D synchronisées sur front descendant. Compléter le schéma en rajoutant un Reset (manuel avec un interrupteur et automatique au démarrage avec une résistance et un condensateur).
   Réaliser un décompteur asynchrone modulo 11 avec des bascules JK synchronisées sur front descendant.

2) Le schéma ci-dessous est celui d'un compteur référencé 74LS93, à partir duquel on peut réaliser des compteurs de différents modulos.



   a) Quel état faut-il appliquer sur R_1 et R_2 pour remettre le compteur à 0 ?
   b) A partir de ce composant, réaliser des compteurs modulo 12 puis 7 puis 13 en rajoutant éventuellement la ou les portes nécessaires.

3) Le schéma ci-dessous est celui d'un compteur référencé 74LS92
   (les entrées de Reset, inutiles pour cet exercice, n'ont pas été représentées)

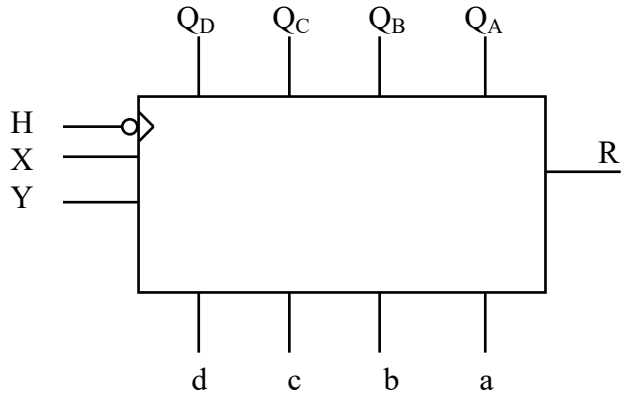


   Tracer le chronogramme de Q_B et de Q_C en fonction d'un signal d'horloge appliqué en H_B
   Quel est le rapport des fréquences f_{Q_D}/f_{H_B} ?
   Si on relie Q_A à H_B et qu'on applique un signal d'horloge en H_A, quel est le rapport f_{Q_D}/f_{H_A} ?

4) On veut réaliser un compteur/décompteur asynchrone modulo 10 avec possibilité de précharger un nombre N = dcba ($0 \leq N \leq 9$).

Ce compteur comportera donc 2 entrées de commande X et Y :

X = 0 → préchargement de N (quelque soit Y)
X = 1 → comptage si Y = 0 et décomptage si Y = 1



On utilisera 4 bascules JK synchronisées sur front descendant et disposant d'entrées de forçage (Reset et Preset) actives à l'état 0 plus les portes logiques nécessaires.

a) Dans un premier temps, dessiner le schéma de connexion des bascules entre elles de manière à obtenir un compteur si Y = 0 et un décompteur si Y = 1 (sans tenir compte du modulo pour l'instant).

b) Remplir la table de vérité ci-dessous des entrées Reset et Preset des 4 bascules en fonction de X et Y(on adoptera les mêmes conventions que dans l'exercice 1: remise systématique à 0 ou à 1 des sorties Q_i en fonction du modulo).

c) En déduire les équations de ces entrées et les simplifier au maximum.

d) Compléter ce compteur/décompteur par la génération d'une retenue R permettant de chaîner plusieurs compteurs/décompteurs.

	X	Y	R_A	Pr_A	R_B	Pr_B	R_C	Pr_C	R_D	Pr_D	R
Ch^t	0	x									
Cp^t	1	0									
$Décp^t$	1	1									

LES COMPTEURS SYNCHRONES

1) Réaliser les compteurs synchrones suivants avec des bascules JK

Compteur modulo 7

Décompteur modulo 10

Compteur modulo 8 en code Gray

Compteur à défilement (ou glissement) : On veut faire défiler un 1 sur les 4 sorties DCBA selon le cycle suivant : 0000 ; 0001 ; 0010 ; 0100 ; 1000 ; 0000 etc …

Compteur à accumulation : On veut "empiler" des 1 sur les 4 sorties DCBA selon le cycle suivant : 0000 ; 0001 ; 0011 ; 0111 ; 1111 ; 0000 etc …

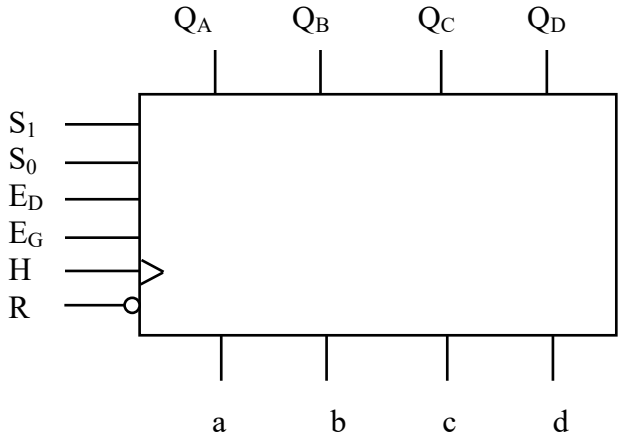
2) Réaliser un compteur modulo 7 avec des bascules D et comparer les 2 montages.

REALISATION D'UN REGISTRE A DECALAGE

Ce registre doit comporter :

- 4 bascules D (notées A, B, C, D de gauche à droite et du poids faible vers le poids fort).
- 2 entrées de commande S_1 et S_0 permettant le fonctionnement selon le tableau décrit ci-dessous.
- 1 entrée série (E_D) utilisée dans le mode décalage à droite.
- 1 entrée série (E_G) utilisée dans le mode décalage à gauche.
- 4 entrées parallèles : a (poids faible), b, c, d (poids fort).
- 1 entrée d'horloge.
- 1 entrée de Reset asynchrone et active à l'état bas

On peut le représenter par le schéma suivant :



Fonctions selon les valeurs de S_1 et de S_0

S_1	S_0	H	fonction
0	0	↑	état mémoire : $Q_A = q_A$ etc…
0	1	↑	décalage à droite : $E_D \rightarrow A \rightarrow B \rightarrow C \rightarrow D$
1	0	↑	décalage à gauche : $E_G \rightarrow D \rightarrow C \rightarrow B \rightarrow A$
1	1	↑	chargement parallèle : $Q_A = a$ etc…

a) Déterminer les équations des entrées D de chaque bascule.

b) Proposer un montage qui permette de faire un Reset soit automatique à l'allumage (utiliser la charge d'un condensateur à travers une résistance) soit manuel avec en plus un bouton poussoir.

INTRODUCTION AU TRAITEMENT SEQUENTIEL

On veut réaliser un additionneur de 2 nombres de 4 bits.

Pour cela on dispose des composants suivants :

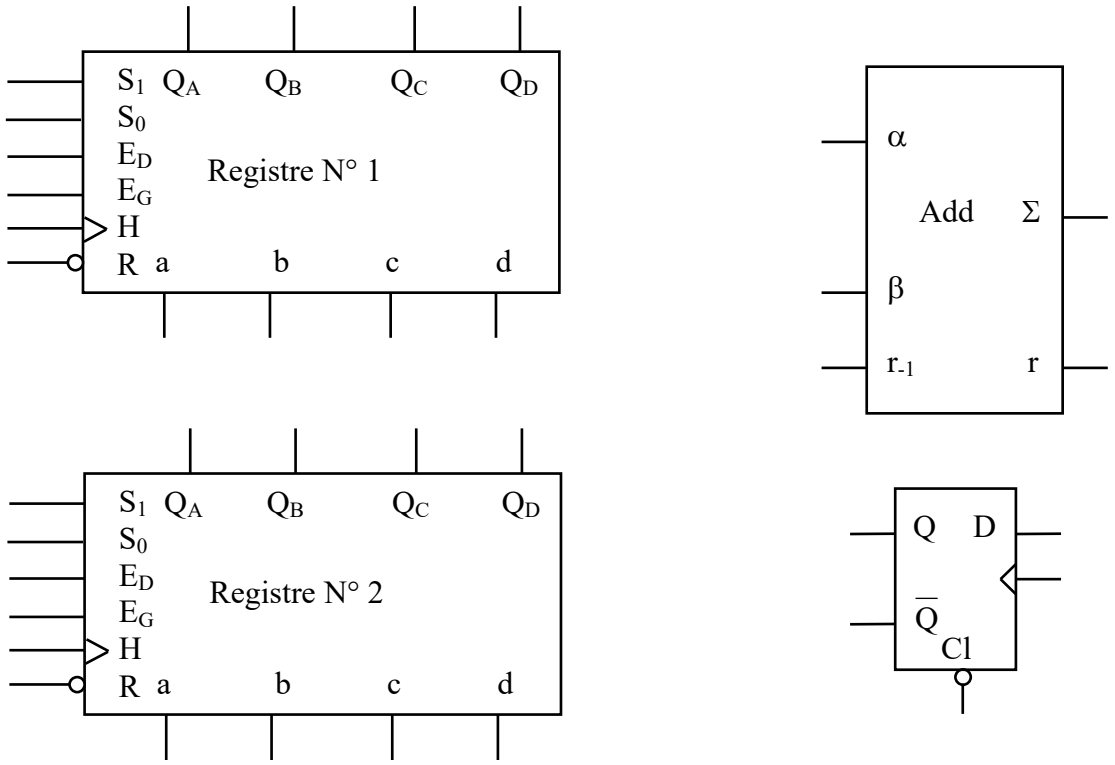
- 1 additionneur de 2 nombres de 1 bit (entrées α, β et r_{-1}, sorties Σ et r)
- 2 registres à décalage (vus au TD précédent)
- Un bascule D avec Reset (asynchrone et actif à l'état bas)
- Les portes logiques éventuellement nécessaires

En début de cycle, le nombre X (x_3, x_2, x_1, x_0) est supposé présent aux entrées de chargement du registre N°1 et Y(y_3, y_2, y_1, y_0) aux entrées de chargement du registre N° 2

Une commande I(nit), active à l'état haut, doit permettre le chargement des registres et l'initialisation des circuits.

En fin de cycle, on doit retrouver X dans le registre N° 1, la somme X + Y dans le registre N° 2 et la retenue éventuelle dans la bascule D.

Compléter le schéma ci-dessous pour obtenir cet additionneur.



NUMERATION : CHANGEMENTS DE BASE

1) 324 ; 1579 ; 185 ; 65 ; 189 ; 2988 ; 4314
2) 1100 0010 ; 1111 1110 ; 401 ; 107A ; 2322
3) 1011 0111 1010 1110 ; 1101 1100 1011 1111 ; 101 0110 0011 0100 1001 ; 101 101 110 111
   125715 ; 43C ; E83A ; 164072
4) 4 ; 12 ; 12
5) a = 10 et b = 5 ; a = 4 et b = 2 ; a = 27 et b = 3
6) base paire : si le chiffre des unités est pair ; base impaire : si la somme des chiffres est paire
7) 13,375 ; 501,53125 ; 4012,61328125
8) 16,514 ; 101010,0110101 ; 45,3AE ; 6C6, 2D8 ; 1364,734

OPERATIONS EN DIFFERENTES BASES

1) 1 0111 1000 ; 1001 1000 ; 1 1000 1011 ; 1360 ; 1176 ; 18294 ; C2DE ; 14E5
2) 101 1010 1110 ; 10010 ; - 101
3) 1010 1010 0101 ; 10 0010 0100 0110
4) 1000,11001 ; 11010,1001
5) a) 2 ; 4 ; 8 ; 2^n b) 2^{13} ; 1FFF ; 2000h c) 2^{15} ; 7FFF ; 8000h d) M_1 : 0000 à 1FFF ; M_2 : 2000
   à 3FFF ; M_3 : 4000 à BFFF ; M_4 : C000 à 13FFF ; capacité : 14000h ; 17 fils
6) a) 1501 valeurs différentes : 11bits b) 150/2047 = 0,073278 mm c) l / 150 = x / 2047
   d) 100 / 0,073278 = 1365_{10} = 555_h = 101 0101 0101 e) 134,538 mm

NOMBRES SIGNES ET CODES

1) a) 1111 1111 ; 1110 0011 ; 1101 1000 ; 1000 0001 ; 1000 0000
2) b) 103 ; erreur ; - 69 ; erreur ; 4
3) b) 111011_G : 1010110_G ; 1000100_G ; c) 136 ; 204 ; 254
4) 1001 0111 ; 0111 0010 0100 ; 0010 0000 0000 1001 ; 0110 0011 0000 1000

LES PORTES LOGIQUES EN ELECTRONIQUE

1)

V_B	V_A
0	0
0	5
5	0
5	5
type de porte	

état de D_B	état de D_A	I_B	I_A	I_R	V_S
B	B	0	0	0	0
B	P	0	1	1	5
P	B	1	0	1	5
P	P	0,5	0,5	1	5
OU					

état de D_B	état de D_A	I_B	I_A	I_R	V_S
P	P	0,5	0,5	1	0
P	B	1	0	1	0
B	P	0	1	1	0
B	B	0	0	0	5
ET					

2)

V_e	état du transistor	V_S
0	Bloqué	5
5	Saturé	0

FONCTIONS LOGIQUES ET SIMPLIFICATION -1

1) a) $A.B.C.D = 1$ ou $\overline{A} + \overline{B} + \overline{C} + \overline{D} = 0$ b) $A + B + C + D = 0$ ou $\overline{A}.\overline{B}.\overline{C}.\overline{D} = 1$
   c) $A + B + C + D = 1$ ou $\overline{A}.\overline{B}.\overline{C}.\overline{D} = 0$ d) $A.B.C.D = 0$ ou $\overline{A} + \overline{B} + \overline{C} + \overline{D} = 0$

2) $S_1 = A.\overline{B} + \overline{A}.B = A \oplus B$ $S_2 = \overline{A} + B = \overline{A.\overline{B}}$ $S_3 = A + C$

$S_4 = A.B + C + D$ $S_5 = A(B + \overline{C})$ $S_6 = \overline{B}(A + \overline{C})$

3) $\overline{S_1} = A.B + \overline{A}.\overline{B} = \overline{A \oplus B}$ $\overline{S_5} = \overline{A} + \overline{B}.C$ $\overline{S_6} = B + \overline{A}.C$

5) $S_1 = \overline{\overline{A.\overline{B}}.\overline{\overline{A}.B}} = \overline{\overline{A} + B + \overline{A} + \overline{B}}$ $S_5 = \overline{\overline{A.\overline{B}}.C} = \overline{\overline{A} + B + \overline{C}}$ $S_6 = \overline{\overline{B}.\overline{A}.C} = \overline{B + \overline{A} + \overline{C}}$

6) $S_1 = A(C + B.D)$ $S_2 = A + C$ $S_3 = B$

FONCTIONS LOGIQUES ET SIMPLIFICATION – 2

3) a) $S = C(B + A)$ b) $S = \overline{C}(\overline{B} + \overline{A})$ c) $S = C \oplus B.A$

4) $A' = A$ $B' = B \oplus A$ $C' = C \oplus (B + A)$

5) $C' = C$ $B' = B \oplus C$ $A' = B \oplus A$

6) $S = D(C + B)$ $S = \overline{A}(\overline{B} + D)$ $S = \overline{A}.\overline{C}$

7) $S = \overline{D}.\overline{C}.\overline{B} + E.D.B + D.\overline{C}.A + E.\overline{D}.\overline{B} + \overline{E}.D.A$ $S = \overline{C}.\overline{A} + D.\overline{B} + D.C.A$

OPERATIONS ARITHMETIQUES

1) $B' = D.\overline{A} + \overline{D}.\overline{C}.A + C.B.\overline{A}$ $C' = D.\overline{A} + A.B + \overline{C}.B$ $D' = D.A + C.\overline{B}.\overline{A}$ $E' = D + C.A + C.B$

2) $S_i = R_{i-1} \oplus (A_i \oplus B_i)$ $R_i = B_i.A_i + R_{i-1}(B_i + A_i)$

3) $D_i = R_{i-1} \oplus (A_i \oplus B_i)$ $R_i = B_i.\overline{A}_i + R_{i-1}(B_i + \overline{A}_i)$

COMPARAISON ET AFFICHAGE

3) $a = d = f = \overline{I}$ $b = \overline{S + E}$ $c = \overline{E}$ $e = \overline{S + I}$ $g = \overline{b}$ soit 5 NOR en tout

4) $a = D + B + \overline{C}.\overline{A} + C.A$ $b = \overline{C} + \overline{B}.\overline{A} + B.A$ $c = \overline{B} + C + A$ $d = D + B.\overline{A} + \overline{C}.B + \overline{C}.\overline{A} + C.\overline{B}.A$
$e = B.\overline{A} + \overline{C}.\overline{A}$ $f = D + \overline{B}.\overline{A} + C.\overline{B} + C.\overline{A}$ $g = D + C.\overline{B} + B.\overline{A} + B.\overline{C}$ ou $D + C.\overline{B} + C.\overline{A} + B.\overline{C}$

MULTIPLEXAGE, DECODAGE ET DEMULTIPLEXAGE

2) a) $S = \overline{A \oplus (B + C)}$ c) 2 puissance 2 puissance n

3) S = 1 si une seule entrée = 1

DECODAGE D'ADRESSES (SIMPLIFIE) D'UN SYSTEME INFORMATIQUE

a) 8 k@ = 2^{13} = 2000_h. Les 13 fils de A_0 à A_{12} sont donc communs à tous les composants.
d) A_{15}, A_{14} et A_{13}
e) Il faut relier A_{13} à l'entrée A du démultiplexeur, A_{14} à l'entrée B et A_{15} à l'entrée C.
f) 960k@; A_{16}, A_{17}, A_{18}, A_{19}

			0 à F	0 à F			000 à FFF
		adresses basses et hautes	A_{19} à A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11} à A_0
EPROM 1	@ basse	0000	0	0	0	0	0	0
	@ haute	01FFF	0	0	0	0	1	1
EPROM 2	@ basse	02000	0	0	0	1	0	0
	@ haute	03FFF	0	0	0	1	1	1
RAM 1	@ basse	04000	0	0	1	0	0	0
	@ haute	05FFF	0	0	1	0	1	1
RAM 2	@ basse	06000	0	0	1	1	0	0
	@ haute	07FFF	0	0	1	1	1	1
RAM 3	@ basse	08000	0	1	0	0	0	0
	@ haute	09FFF	0	1	0	0	1	1
RAM 4	@ basse	0A000	0	1	0	1	0	0
	@ haute	0BFFF	0	1	0	1	1	1
DUART	@ basse	0C000	0	1	1	0	0	0
	@ haute	0DFFF	0	1	1	0	1	1
I/0	@ basse	0E000	0	1	1	1	0	0
	@ haute	0FFFF	0	1	1	1	1	1

UNITE LOGIQUE

S_3	S_2	S_1	S_0	X(A, B)	Y(A, B)	F(A, B)
0	0	0	0	1	$\overline{A}$	$\overline{A}$
0	0	0	1	1	$\overline{A+B}$ ou $\overline{A}.\overline{B}$	$\overline{A+B}$ ou $\overline{A}.\overline{B}$
0	0	1	0	1	$\overline{A}.B$ ou $\overline{A+\overline{B}}$	$\overline{A}.B$ ou $\overline{A+\overline{B}}$
0	0	1	1	1	0	0
0	1	0	0	$\overline{A}+B$ ou $\overline{A.\overline{B}}$	$\overline{A}$	$\overline{A.B}$ ou $\overline{A}+\overline{B}$
0	1	0	1	$\overline{A}+B$ ou $\overline{A.\overline{B}}$	$\overline{A+B}$ ou $\overline{A}.\overline{B}$	$\overline{B}$
0	1	1	0	$\overline{A}+B$ ou $\overline{A.\overline{B}}$	$\overline{A}.B$ ou $\overline{A+\overline{B}}$	$A \oplus B$
0	1	1	1	$\overline{A}+B$ ou $\overline{A.\overline{B}}$	0	$\overline{\overline{A}+B}$ ou $A.\overline{B}$
1	0	0	0	$\overline{A.B}$ ou $\overline{A}+\overline{B}$	$\overline{A}$	$\overline{A}+B$ ou $\overline{A.\overline{B}}$
1	0	0	1	$\overline{A.B}$ ou $\overline{A}+\overline{B}$	$\overline{A+B}$ ou $\overline{A}.\overline{B}$	$\overline{A \oplus B}$
1	0	1	0	$\overline{A.B}$ ou $\overline{A}+\overline{B}$	$\overline{A}.B$ ou $\overline{A+\overline{B}}$	B
1	0	1	1	$\overline{A.B}$ ou $\overline{A}+\overline{B}$	0	A.B
1	1	0	0	$\overline{A}$	$\overline{A}$	1
1	1	0	1	$\overline{A}$	$\overline{A+B}$ ou $\overline{A}.\overline{B}$	$A+\overline{B}$ ou $\overline{\overline{A}.B}$
1	1	1	0	$\overline{A}$	$\overline{A}.B$ ou $\overline{A+\overline{B}}$	A + B
1	1	1	1	$\overline{A}$	0	A

LES BASCULES R S ASYNCHRONES

1) b) 0 donc arrêt prioritaire
   c) l'une des sorties revient à 1 mais on ne sait pas laquelle : l'état interdit est donc l'état précédent (celui où R = S = 1) car on risque de ne pas connaître pas l'état des sorties après cet état.

2) b) 1 donc marche prioritaire
   c) même réponse que ci-dessus (en remplaçant 1 par 0 pour les sorties)

3) a) marche prioritaire pour maintenir l'état alarme quand on appuie sur Reset alors qu'un des capteurs reste activé.

LES BASCULES R S SYNCHRONES

2) C'est une bascule maître-esclave, synchronisée sur impulsion (voir le cours page 31)
4) C'est un diviseur de fréquence par 2.
5) compteur modulo 3 ; décompteur modulo 4 ; décompteur modulo 3

LES BASCULES D

2) C'est un diviseur de fréquence par 2.
4) compteur modulo 3 ; compteur modulo 4 ; compteur modulo 5

LES BASCULES JK

2) C'est un diviseur de fréquence par 2.
3) compteur modulo 3 ; décompteur modulo 4 ; compteur modulo 4 en code Gray
5) Générateur de rafales d'impulsions synchronisées sur l'horloge.
6) Générateur de simple impulsion de durée calibrée (transfert en mémoire).

LES COMPTEURS ASYNCHRONES

2) a) 1
3) diviseur de fréquence par 6 puis par 12.
4) $R_A = \overline{X}.a + X.(Y + K)$; $P_A = \overline{X}.\overline{a} + X.(\overline{Y} + K)$; $R_B = \overline{X}.b + X.K$; $P_B = X + \overline{b}$; $Ret = \overline{X} + K$

LES COMPTEURS SYNCHRONES

	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
Compteur mod 7	$\overline{B.C}$	1	A	A + C	A.B	B		
Décompteur mod 10	1	1	$\overline{A}.(C + D)$	$\overline{A}$	$D.\overline{A}$	$\overline{A + B}$	$\overline{A + B + C}$	$\overline{A}$
Compteur mod 8 en Gray	$\overline{C \oplus B}$	$C \oplus B$	$\overline{C}.A$	C.A	$B.\overline{A}$	$\overline{A + B}$		
Compteur à défilement	$\overline{D}.\overline{C}.\overline{B}$	1	A	1	B	1	C	1
Compteur à accumulation	1	D	A	D	B	D	C	1

REGISTRE A DECALAGE

$D_A = \overline{S}_1.\overline{S}_0.Q_A + \overline{S}_1.S_0.E_D + S_1.\overline{S}_0.Q_B + S_1.S_0.a$ $\quad D_B = \overline{S}_1.\overline{S}_0.Q_B + \overline{S}_1.S_0.Q_A + S_1.\overline{S}_0.Q_C + S_1.S_0.b$

$D_C = \overline{S}_1.\overline{S}_0.Q_C + \overline{S}_1.S_0.Q_B + S_1.\overline{S}_0.Q_D + S_1.S_0.c$ $\quad D_D = \overline{S}_1.\overline{S}_0.Q_D + \overline{S}_1.S_0.Q_C + S_1.\overline{S}_0.E_G + S_1.S_0.d$