ECOLE CENTRALE D'ELECTRONIQUE

GROUPE ECE

Circuits logiques et électronique numérique

- Support de cours -

COURS ING3
Année 2007-2008

Benoît Decoux

Sommaire

Introduction générale

Bienvenue dans le monde merveilleux de la logique et de l'électronique numérique, dans lequel on a :

$$1+1=1$$

mais aussi :

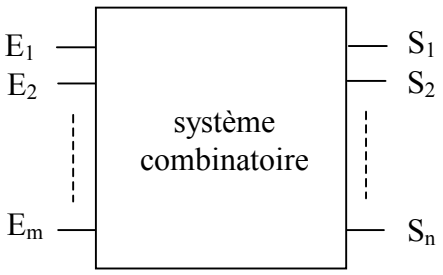
$$1+1=10$$

…

Dans le domaine de la logique, on distingue en général 2 grandes catégories :

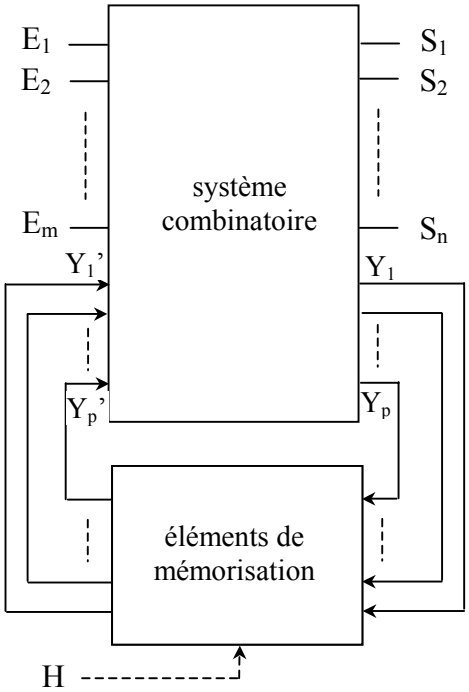
- la logique combinatoire,
- la logique séquentielle.

Dans les circuits combinatoires, les sorties sont déterminées uniquement en fonction des variables d'entrée. Le temps n'intervient pas dans les fonctions logiques.



Système séquentiel à m entrées (E_1, E_2, …, E_m), n sorties (S_1, S_2, …, S_n)

Dans un système séquentiel, l'état de sortie ne dépend pas uniquement de la combinaison des entrées à un instant donné, mais aussi de valeurs passées d'entrées, de sorties ou de variables internes. Dans notre cas, il s'agira de valeurs précédentes des sorties. En plus de dépendre des entrées présentes, les sorties présentes seront exprimées en fonction de leur valeur précédente, disponible grâce à la présence d'éléments de mémorisation. Les systèmes séquentiels sont donc des systèmes bouclés. Le schéma suivant illustre ces propriétés :



Système séquentiel à m entrées (E_1, E_2, …, E_m), n sorties (S_1, S_2, …, S_n) et p variables internes (Y_1', Y_2',…, Y_p')

Sur ce schéma, Y_1', Y_2', ...,Y_p' représentent les valeurs présentes de variables internes et Y_1, Y_2, ...,Y_p leur valeur passées. La notion de présent et de passé dépend du type de système séquentiel utilisé : il existe les système séquentiels sychrones et les systèmes séquentiels asynchrones.

Dans les systèmes séquentiels **synchrones**, c'est un signal d'horloge (une alternance, dans le temps, de 0 et de 1) qui définit le passage du passé au présent : la mise à jour des variables présentes Y_1', Y_2', ...,Y_p' se produit à chaque "coup" d'horloge. Ce signal est représenté en pointillé sur le schéma ci-dessus.

Dans les systèmes séquentiels **asynchrones**, les éléments de mémorisation sont constitués par un retour direct. La mise à jour des variables présentes Y_1', Y_2', ...,Y_p' est donc quasi-instantanée après la mise à jour des variables passées Y_1, Y_2, ...,Y_p.

Par exemple, un compteur est un système séquentiel qui ne possède comme entrée qu'un signal d'horloge. A chaque coup d'horloge, sa valeur de sortie binaire s'incrémente. A un instant donné, sa sortie dépend de sa valeur au coup d'horloge précédent.

Plan du cours

Le cours sera subdivisé en 4 grandes parties :

1) Fonctions logiques de base et circuits associés :
   - Fonctions simples : NON, ET et OU, et fonctions composées : fonctions NON-ET, NON-OU, OU EXCLUSIF, et opérateurs logiques correspondant.
   - Algèbre de Boole
2) Logique combinatoire :
   - Codeur/décodeur, transcodeur, multiplexeur, additionneur…
3) Logique séquentielle :
   - Bascules, registres, compteurs…
4) Familles de circuits intégrés logiques :
   - Etude des différences entre les familles TTL et CMOS.

Du point de vue électronique…

Concrètement, lors de la réalisation de circuits électroniques numériques, les 2 niveaux logiques sont constitués par 2 tensions différentes. La tension correspondant au niveau 0 est en général 0V. La tension correspondant au niveau 1 dépend de la technologie utilisée. Une norme couramment répandue est la norme TTL :

niveau logique 0 ↔ 0 Volts
niveau logique 1 ↔ 5 Volts

Les opérateurs logiques de base et d'autres fonctions logiques plus évoluées existent sous forme de circuits intégrés.

Vocabulaire

Dans ce cours, un vocabulaire spécifique au domaine de la logique sera utilisé.

Le **bit** est l'élément de base de la logique binaire : il vaut 0 ou 1. C'est l'équivalent du chiffre dans le domaine décimal (0 à 9).

Un **mot** binaire est composé d'un nombre variables de bits. On peut également parler de **nombre binaire**.

En général, la logique utilisée est la **logique positive**, dans laquelle le niveau dit actif est le niveau 1. Ca sera le cas dans la totalité de ce cours. Mais il existe également la logique négative, dans laquelle il s'agit du 0.

Partie I) Fonctions logiques de base et circuits associés

Cette partie abordera la représentation des fonctions sous forme algébrique, sous forme de table de vérité puis sous forme de shémas, et leur simplification au moyen des règles de l'algèbre de Boole et des tableaux de Karnaugh.

I.1) Algèbre de Boole

Georges Boole, philosophe et mathématicien irlandais du 19ème siècle est l'auteur d'une théorie sur l'art de construire un raisonnement logique au moyen de propositions qui ont une seule réponse OUI (VRAI) ou NON (FAUX). L'ensemble des opérations formelles appliquées à ces propositions forme une structure mathématique appelée algèbre de Boole. A son époque, il s'agissait de développement purement théorique car on ignorait l'importance qu'allait prendre cette algèbre avec l'informatique.

Les concepts de la logique booléenne ont été ensuite appliqués aux circuits électroniques par Claude Shannon (1916-2001).

Cette algèbre est applicable à l'étude des systèmes possédant deux états s'excluant mutuellement. Dans la logique positive (la plus couramment utilisée), on associe OUI à 1 et NON à 0. Dans la logique négative, c'est l'inverse.

L'algèbre booléenne binaire est à la base de la mise en œuvre de tous les systèmes numériques : ordinateurs, systèmes numériques portables, systèmes de communication numériques, etc. Elle permet entre autres de simplifier les fonctions logiques, et donc les circuits électroniques associés.

I.1.1) Fonctions logiques de base et opérateurs correspondant

Une fonction logique est une fonction d'une ou plusieurs variables logiques, combinées entre elles par 3 fonctions élémentaires simples : NON, OU et ET.

Il existe également des fonctions élémentaires composées de fonctions élémentaires simples : NON-ET, NON-OU, OU-EXCLUSIF, NON-OU-EXCLUSIF.

Elles peuvent être représentées schématiquement pas des opérateurs logiques, encore appelés portes logiques.

a) Fonctions élémentaires simples : NON, OU, ET

Fonction NON (ou fonction complément)

Soit A une variable quelconque. La fonction complément de la variable A est notée :

$$F(A) = \overline{A}$$

(prononcer "a barre", "non A", ou encore "complément de A").

Remarque : une variable est également appelée "littéral".

Cette fonction affecte à la variable de sortie l'état complémentaire de la variable d'entrée.

Table de vérité

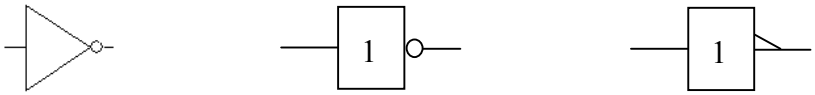
On peut exprimer cette propriété sous forme d'un tableau entrée/sortie, appelé table de vérité :

A	F(A)
0	1
1	0

Le nom anglais de la fonction complément est "NOT".

Symboles

Il existe deux types de symbole pour représenter cet opérateur. Le symbole américain, très utilisé dans les logiciels de simulation (à gauche sur la figure ci-dessous), et les symboles européens, normalisés (au milieu et à droite).



Le cercle est en général utilisé pour indiquer une complémentation.
On appelle souvent cet opérateur "inverseur".

Fonction OU (ou somme logique)

La fonction logique OU est également appelée "somme logique", ou "union logique". Sa notation algébrique utilise le symbole de la somme arithmétique. Pour 2 variables A et B, on a :

$$F(A, B) = A+B$$

Le terme anglais est "OR".

Table de vérité

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Symboles

Remarque 1

On verra plus loin que la somme logique est différente de la somme arithmétique. Dans le cas de la fonction OU, on a :

$$1+1=1$$

Avec l'opération arithmétique +, on aurait :

$$1+1=10$$

(en d'autres termes, le résultat est 0 avec une retenue égale à 1).

Remarque 2

On pourrait avoir plus de 2 variables. Par exemple, pour 3 variables :

A	B	C	A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Le symbole correspondant serait le même mais avec 3 entrées.

Fonction ET (ou produit logique)

La fonction ET est également appelée "produit logique", ou "intersection logique".
Sa notation algébrique utilise le symbole de la multiplication arithmétique :

$$F(A, B) = A \times B$$

ou encore :

$$F(A, B) = A.B$$

ou plus simplement

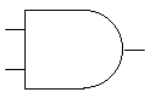
$$F(A, B) = AB$$

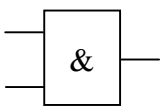
Le terme anglais est "AND".

Table de vérité

A	B	A×B
0	0	0
0	1	0
1	0	0
1	1	1

Symboles





Remarque

Avec la fonction ET à plus de 2 entrées, le seul cas où la sortie serait à 1 serait le cas où toutes les entrées sont à 1.

Illustration

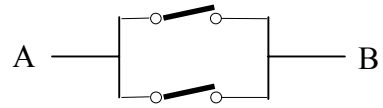
Pour se souvenir des fonctions OU et ET, on peut utiliser une analogie électrique : des interrupteurs fermés ou ouverts et donc laissant passer le courant ou non :

- interrupteurs en parallèle pour le OU ;
- interrupteurs en série pour le ET.

La correspondance entre l'état de l'interrupteur et le niveau logique est :

ouvert (le courant ne passe pas) ↔ niveau logique 0

fermé (le courant passe) ↔ niveau logique 1



Le courant passe de A à B si l'un OU l'autre des 2 interrupteurs est fermé.

Le courant passe de A à B si l'un ET l'autre des 2 interrupteurs sont fermés.

Exemple de fonction composée de ces 3 fonctions de base

Soit la fonction logique suivante, de 4 variables A, B, C et D :

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

On souhaiterait savoir pour quelles valeurs des variables cette fonction vaut 1.

La fonction étant une somme logique, elle vaut 1 si au moins un des termes de la somme vaut 1. Chaque terme étant un produit logique, il vaut 1 si tous ses termes valent 1, soit si :

$$A = B = C = D = 1$$

$$A = 1; B = 0; C = 0; D = 1$$

$$A = 0; B = 0; C = 0; D = 0$$

Donc, sur les 16 combinaisons possibles des variables d'entrée A, B, C, D, 3 seulement provoquent un 1 en sortie.

b) Fonctions élémentaires composées : NON-ET, NON-OU, OU-EXCLUSIF, NON-OU-EXCLUSIF

Les fonctions élémentaires composées (ou combinées, ou induits) sont obtenues en combinant entre eux les fonctions élémentaires simples NON, ET et OU. L'ensemble des fonctions élémentaires simples et des fonctions élémentaires combinées NON-ET, NON-OU, OU-EXCLUSIF, NON-OU-EXCLUSIF définissent un ensemble complet d'opérateurs.

Fonction NON-OU (NOR)

La fonction NON-OU est obtenue en complémentant la sortie d'un OU, c'est à dire en appliquant la sortie de la fonction OU à la fonction NON. Pour 2 variables A et B, elle est notée :

$$F(A, B) = \overline{A + B}$$

On utilise également la notation (moins courante) :

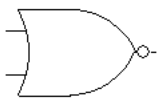
$$F(A, B) = A \downarrow B$$

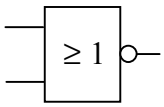
Table de vérité

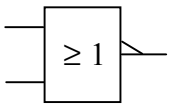
Pour obtenir ta table de vérité du non-OU, il suffit d'inverser les valeurs de sortie dans la table de vérité du OU :

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Symboles







Fonction NON-ET (NAND)

La fonction NON-ET est obtenue en complémentant la sortie d'un ET, c'est à dire en appliquant la sortie de la fonction ET à la fonction NON. Pour 2 variables A et B, elle est notée :

$$F(A, B) = \overline{A.B}$$

On utilise également la notation :

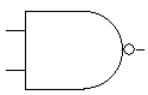
$$F(A, B) = A \uparrow B$$

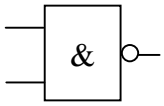
Table de vérité

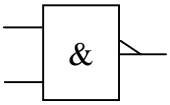
Pour obtenir la table de vérité du non-ET, il suffit d'inverser les valeurs de sortie dans la table de vérité du ET :

A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Symboles







Fonction OU EXCLUSIF (XOR)

Pour 2 variables A et B, la fonction OU EXCLUSIF est définie par :

$$F(A, B) = A\overline{B} + \overline{A}B$$

On la note également :

$$F(A, B) = A \oplus B$$

Table de vérité

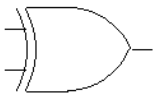
Pour 2 variables, le OU EXCLUSIF vaut 1 si une et une seule des deux variables d'entrée est à 1.

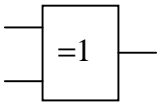
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Avec 2 variables, on peut considérer le OU-EXCLUSIF comme un détecteur d'inégalité, puisque sa sortie est à 1 quand ses 2 entrées sont différentes.

Pour un nombre supérieur de variables, la fonction OU-EXCLUSIF vaut 1 quand les variables d'entrée à 1 sont en nombre pair. Il s'agit donc également d'un détecteur de parité.

Symboles utilisés





Fonction NON-OU EXCLUSIF (XNOR)

La fonction NON-OU EXCLUSIF est obtenue en complémentant la sortie d'un OU EXCLUSIF, c'est à dire en appliquant la sortie de la fonction OU EXCLUSIF à la fonction NON. Pour 2 variables A et B, elle est notée :

$$F(A, B) = \overline{A \oplus B}$$

On peut montrer qu'elle est égale à :

$$\overline{A \oplus B} = A.B + \overline{A}\,\overline{B}$$

Démonstration (utilise des propriétés étudiées dans le paragraphe suivant) :

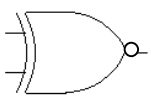
$$\overline{A \oplus B} = \overline{A\overline{B} + \overline{A}B} = \overline{A\overline{B}}\,\overline{\overline{A}B} = (\overline{A} + \overline{\overline{B}}).(\overline{\overline{A}} + \overline{B})$$
$$= (\overline{A} + B).(A + \overline{B}) = \overline{A}A + \overline{A}\overline{B} + B.A + B\overline{B}$$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

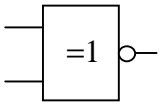
Table de vérité

Pour obtenir la table de vérité du NON-OU EXCLUSIF, il suffit d'inverser les valeurs de sortie dans la table de vérité du OU EXCLUSIF :

A	B	F(A,B)
0	0	1
0	1	0
1	0	0
1	1	1

Symbole





I.1.2) Propriétés et théorèmes

a) Fonctions élémentaires simples NON, OU, ET

Commutativité

$$A+B = B+A$$
$$A.B=B.A$$

Associativité

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

Eléments neutres

$$A+0=A$$
$$A.1=A$$

Eléments absorbants

$$A+1=1$$
$$A.0=0$$

Idempotence (ou redondance)

$$A.A.A…..A=A$$
$$A+A+A+….+A=A$$

Propriétés de la fonction complément

$$\overline{\overline{A}} = A$$

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

$$A.\overline{A} = 0$$

Théorème de DeMorgan

Le complément d'une somme est égal au produit des termes complémentés :

$$\overline{A + B} = \overline{A}\,\overline{B}$$

Le complément d'un produit est égal à la somme des termes complémentés :

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

Cette propriété est généralisable à n variables.

Double distributivité

Distributivité du produit par rapport à la somme :

$$A.(B+C)=AB+AC$$

Distributivité de la somme par rapport au produit :

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

De la droite vers la gauche du signe égal, on parle de factorisation.

La 2^{e} propriété peut sembler étonnante : c'est la seule propriété de ce paragraphe qui diffère de celles de l'algèbre traditionnelle. On pourra pourtant vérifier qu'elle est exacte, en utilisant les propriétés d'idempotence et d'élément neutre :

$$\begin{aligned}(A+B).(A+C) &= A.A+A.C+B.A+B.C\\ &=A+A.C+B.A+B.C\\ &=A.(1+C)+B.A+B.C\\ &=A+B.A+B.C\\ &=A.(1+B)+B.C\\ &=A+B.C\end{aligned}$$

Absorption

Lorsqu'une somme logique contient un terme et un de ses multiples, on peut négliger le multiple :

$$A+A.B = A$$

On peut constater que cette règle reste vraie si on intervertit les opération ET et OU :

$$A.(A+B) = A$$

Absorption du complément

1) Absorption du complément dans une somme :

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

Démonstration :

$$\begin{aligned} A + B &= (A + \overline{A}.)(A + B) \\ &= AA + \overline{A}A + AB + \overline{A}B \\ &= A + \overline{A}A + AB + \overline{A}B \\ &= A + AB + \overline{A}B \\ &= A.(1 + B) + \overline{A}B \\ &= A + \overline{A}B \end{aligned}$$

De la même manière :

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

2) Absorption du complément dans un produit :

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

Démonstration :

$$\begin{aligned} A.(\overline{A} + B) &= A\overline{A} + AB \\ &= AB \end{aligned}$$

Principe de dualité

Le principe de dualité énonce que :

"Toute expression logique vraie demeure vraie si on remplace les + par des . , les 0 par des 1 et les 1 par des 0"

Exemples :

$$A + 1 = 1 \quad \leftrightarrow \quad A.0 = 0$$

$$A + \overline{A}B = A + B \quad \leftrightarrow \quad A(\overline{A} + B) = AB$$

$A + B = 1$: vrai si A=1 ou B=1 $\quad \leftrightarrow \quad$ $A.B = 0$: vrai si A=0 ou B=0

Théorème des consensus

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

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

Le 2^{e} est le dual du 1er ; on peut donc l'obtenir par application du principe de dualité.

b) Fonctions élémentaires composées NON-OU, NON-ET, OU EXCLUSIF, NON-OU EXCLUSIF

Commutativité

Le NON-OU et le NON-ET, le OU EXCLUSIF et le NON-OU EXCLUSIF sont commutatifs.

Associativité

Le NON-OU et le NON-ET ne sont pas associatifs :

$$\overline{\overline{A.B}.C} \neq \overline{A.\overline{B.C}}$$

$$\overline{\overline{A+B}+C} \neq \overline{A+\overline{B+C}}$$

Le OU EXCLUSIF et le NON-OU EXCLUSIF sont associatifs.

Vérification de l'associativité du OU EXCLUSIF en étudiant le cas de 3 variables. En calculant d'abord $A \oplus B$:

A	B	$A \oplus B$	C	$A \oplus B \oplus C$
0	0	0	0	0
0	0	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1

Recommençons en calculant d'abord $B \oplus C$:

A	B	C	$B \oplus C$	$A \oplus B \oplus C$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

On constate donc que le OU EXCLUSIF est associatif :

$$A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$$

A	B	$\overline{A \oplus B}$	C	$\overline{\overline{A \oplus B} \oplus C}$
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
1	0	0	0	1
1	0	0	1	0
1	1	1	0	0
1	1	1	1	1

A	B	C	$\overline{B \oplus C}$	$\overline{A \oplus \overline{B \oplus C}}$
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Le NON-OU EXCLUSIF est donc associatif.

Autres propriétés du OU-EXCLUSIF

On remarque que l'on a la propriété suivante :

$$A \oplus A = 0$$

L'élément neutre du OU-EXCLUSIF est 0.

I.2) Représentation des fonctions logiques

Il existe plusieurs manières de représenter une fonction logique : par
- table de vérité,
- expression algébrique,
- schéma à portes logiques.

On peut passer facilement d'une représentation à l'autre. La connaissance de l'une des 3 est suffisante à la connaissance totale de la fonction.

I.2.1) Représentation par table de vérité

Une fonction F de n variables est entièrement décrite par l'énoncé de l'ensemble des combinaisons des variables d'entrées et de la valeur de la fonction correspondant à chaque combinaison. Cet énoncé prend généralement la forme d'un tableau à n+1 colonnes (n entrées + 1 sortie) et 2^n lignes (sur n bits, on peut coder 2^n valeurs différentes).

Chaque ligne comporte donc une combinaison des variables, qui valent 0 ou 1, et la valeur correspondante de la fonction F(A,B,C) qui vaut également 0 ou 1.

A	B	C	F(A,B,C)
0	0	0	F(0,0,0)
0	0	1	F(0,0,1)
0	1	0	F(0,1,0)
0	1	1	F(0,1,1)
1	0	0	F(1,0,0)
1	0	1	F(1,0,1)
1	1	0	F(1,1,0)
1	1	1	F(1,1,1)

Remarque 1

Une table de vérité peut comporter plusieurs colonnes de sorties (par exemple, celle qui définirait le passage du code binaire naturel au code de Gray). Dans ce cas, il y a une fonction pour chaque sortie.

Remarque 2

Selon l'application, il peut y avoir des combinaisons d'entrée non-utilisées. Sous certaines conditions, on peut alors utiliser ces combinaisons pour simplifier la fonction (voir tableaux de Karnaugh, plus loin).

Exemple

On définit la fonction logique f (A, B, C) = 1 si $(A, B, C)_2 > 5$. La table de vérité correspondante est :

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

I.2.2) Représentation algébrique

Il s'agit d'une représentation sous forme d'expression.

Une fonction logique Booléenne se présente comme une association d'opérations algébriques sur un ensemble de variables logiques. Elle peut s'exprimer comme une association de sommes et de produits logiques.

a) Ecriture sous forme d'une somme de produits (SDP)

Cette forme est également appelée forme **disjonctive**. Par exemple, la fonction :

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

est sous forme conjonctive.

Pour mettre une fonction sous cette forme, on peut utiliser les règles de l'algèbre de Boole, et notamment la distributivité du produit par rapport à la somme.

Exemple

$$(A+B)(B+C+D) = AB + AC + AD + BB + BC + BD$$
$$= AB + AC + AD + B + BC + BD$$

Forme SDP standard

Dans la **forme standard**, chacun des termes de la somme doit comporter toutes les variables.

On dit également que la fonction est sous forme **normale** ou **canonique**.

Sous cette forme standard, chacun des termes de la somme est appelé **minterme**.

Si une fonction n'est pas sous forme standard, on peut faire apparaître les variables manquantes. Par exemple, dans la fonction :

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

il manque la variable D dans le 1er terme. Pour le faire apparaître, on peut le multiplier par

$$D + \overline{D}$$

car ce terme vaut 1, et le 1 est l'élément neutre pour le produit logique. On a donc :

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

d'où la fonction :

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

L'intérêt de la forme standard est de faciliter l'écriture de la table de vérité et des tableaux de Karnaugh (voir plus loin). Elle permet de remplir directement des derniers.

Détermination de l'expression algébrique à partir de la table de vérité

Pour chaque ligne où la sortie vaut 1, on effectue les produits des variables d'entrée, complémentées si elles valent 0, non complémentées si elles valent 1. Puis on effectue la somme de ces différents produits.

Par exemple, pour la fonction OU :

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

La forme disjonctive de la fonction est :

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

Par utilisation des règles de l'algèbre de Boole, on retrouve bien l'expression de la fonction OU :

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

b) Ecriture sous forme d'un produit de somme (PDS)

Cette forme est également appelée forme **conjonctive**. Par exemple, la fonction :

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

est sous forme disjonctive.

Forme PDS standard

Dans la forme disjonctive standard, chaque terme du produit contient toutes les variables. La fonction de l'exemple ci-dessus n'est pas sous la **forme standard**.

Chacun des termes de la forme PDS standard est appelé **maxterme**.

Pour mettre une fonction sous forme standard, une méthode consiste à faire apparaître les variables manquantes en utilisant la propriété :

$$A.\overline{A} = 0$$

où A est une variable quelconque, et la propriété de distributivité de la somme par rapport au produit (rappel : $A + BC = (A + B).(A + C)$).

A chaque terme du produit auquel il manque des variables, on introduit ces dernières en utilisant cette propriété. Par exemple pour la fonction suivante de 3 variables A, B et C :

$$(A + B)(\overline{A} + B + \overline{C}) = (A + B + 0)(\overline{A} + B + \overline{C}) = (A + B + C.\overline{C})(\overline{A} + B + \overline{C}) =$$
$$(A + B + C)(A + B + \overline{C})(\overline{A} + B + \overline{C})$$

Détermination de la forme PDS à partir de la table de vérité

Pour déterminer l'expression de la fonction sous forme disjonctive standard à partir de la table de vérité, on considère les lignes où la fonction vaut 0 ; chaque 0 de la sortie correspond à un terme du PDS.

On écrit d'abord la forme SDP comme on l'a fait précédemment ; le résultat correspond à $\overline{F}$. On doit alors transformer la SDP en PDS en utilisant les règles de l'algèbre de Boole.

Par exemple, pour la table de vérité du OU EXCLUSIF :

A	B	F(A,B)
0	0	0
0	1	1
1	0	1
1	1	0

On a :

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

d'où

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

On peut remarquer qu'on peut également écrire directement le PDS à partir de la table, en considérant les lignes où F vaut 0, et en complémentant les variables.

Exemple

On cherche à déterminer l'expression algébrique, d'abord sous forme SDP puis sous forme PDS, standard puis simplifiée, de la fonction définie par la table de vérité suivante :

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Forme SDP :

$$F = \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + A.B.C \quad \text{(forme standard)}$$
$$= \overline{A}.\overline{B}.C + A.\overline{B}.(\overline{C} + C) + A.B.(\overline{C} + C)$$
$$= \overline{A}.\overline{B}.C + A.(\overline{B} + B)$$
$$= A + \overline{B}.C \quad \text{(forme simplifiée)}$$

Forme PDS :

1ère méthode :

$$\overline{F} = \overline{A}.B.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C \quad \text{(forme standard)}$$
$$= \overline{A + B + C} + \overline{A + \overline{B} + C} + \overline{A + \overline{B} + \overline{C}}$$
$$= \overline{(A + B + C).(A + \overline{B} + C).(A + \overline{B} + \overline{C})}$$

d'où

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

2e méthode : la détermination est directe :

$$F = (A + B + C).(A + \overline{B} + C).(A + \overline{B} + \overline{C}) \quad \text{(forme standard)}$$

Détermination de la forme PDS à partir de la forme SDP

On peut passer de la forme disjonctive à la forme conjonctive en effectuant des factorisations, et en utilisant la propriété de distributivité de la somme par rapport au produit.

On peut également utiliser une méthode plus systématique, consistant à

- calculer $\overline{F}$ et le mettre sous forme SDP
- lui appliquer le principe de dualité à l'expression obtenue, en :
  - remplaçant F par $\overline{F}$;
  - remplaçant × par + et vice-versa ;
  - complémentant les variables.

Par exemple, avec la fonction OU EXCLUSIF sous forme PDS standard :

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

On a :

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

On applique alors les transformations décrites ci-dessus à cette dernière expression, et on obtient

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

I.2.3) Expression numérique

Pour simplifier la représentation de la fonction, on peut l'exprimer sous forme numérique. Cette forme indique la valeur décimale correspondant aux combinaisons binaires des variables, pour lesquelles la fonction vaut 1.

Par exemple, pour la fonction OU-EXCLUSIF, dont on rappelle la table de vérité :

A	B	F(A,B)
0	0	0
0	1	1
1	0	1
1	1	0

Elle peut être notée :

$$f(A,B,C) = \Sigma\ (1,2)$$

De même, la fonction logique définie par :

$$f\ (A, B, C) = 1 \text{ si } (A, B, C)_2 > 5$$

peut être notée sous la forme :

$$f(A,B,C) = \Sigma\ (6,7)$$

I.3) Simplification des fonctions logiques

En pratique, les fonctions logiques sont réalisées avec des circuits électroniques. Généralement, on cherche à représenter la fonction avec un minimum de termes car une fonction simplifiée utilisera moins de circuits. Elle sera exécutée plus rapidement (une porte logique possède un temps de propagation) et à un moindre coût.

Quand on parle de simplification, on sous-entend qu'une même complémentation (barre de surlignage) ne peut s'étendre sur plus d'une variable. Si ça n'est pas le cas, la fonction peut encore être simplifiée.

I.3.1) Méthode algébrique

On utilise les différents théorèmes et propriétés de l'algèbre de Boole.

Exemple 1

Simplification de la fonction

$$F(A,B,C)=AB+A(B+C)+B(B+C)$$

On a :

$$\begin{aligned} F(A,B,C)&=AB+AB+AC+BB+BC \\ &=AB+AB+AC+B+BC \\ &=AB+AC+B+BC \\ &=AB+AC+B \\ &=AC+B \end{aligned}$$

Exemple 2

$$\begin{aligned} F &= [A.\overline{B}(C+BD)+\overline{A.B}]C \\ &= [A.\overline{B}C + A.\overline{B}BD + \overline{A.B}]C = [A.\overline{B}C + A.0.D + \overline{A.B}]C = [A.\overline{B}C + 0 + \overline{A.B}]C \\ &= [A.\overline{B}C + \overline{A.B}]C = A.\overline{B}CC + \overline{A.B}C = A.\overline{B}C + \overline{A.B}C = (A+\overline{A}.).\overline{B}C \\ &= .\overline{B}.C \end{aligned}$$

I.3.2) Tableaux de Karnaugh

Les tableaux de Karnaugh constituent une autre représentation de la table de vérité de la fonction. Ils permettent de simplifier les fonctions logiques de manière graphique.

Principe

Les variables d'entrée sont placées dans la 1ère case en haut à gauche du tableau. Elle sont réparties en lignes et colonnes. Par exemple, pour 3 variables, on peut en utiliser 2 pour constituer 4 lignes (correspondant aux 4 combinaisons de ces 2 variables) et 1 pour constituer 2 colonnes (correspondant aux 2 valeurs de cette variable).

Pour que la simplification puisse se faire, il faut qu'entre 2 lignes et 2 colonnes adjacentes, 1 seul bit change dans les combinaisons des variables. Cette règle doit être vraie également entre la dernière ligne (ou colonne) et la première. On peut pour cela utiliser le code binaire réfléchi (ou "code de Gray"), qui possède ces caractéristiques.

Par exemple, le code binaire réfléchi pour 2 bits est :

00
01
11
10

puis pour 3 bits :

000
001
011
010
110
111
101
100

Il y a bien un seul bit différent entre la dernière combinaison et la première.

Quand on ajoute une variable supplémentaire (ce qui va avoir pour effet de doubler le nombre de valeurs possibles), on la met à 0 pour les valeurs déjà définies et à 1 pour les nouvelles valeurs. Pour les autres variables, on les recopie par symétrie de leurs valeurs déjà définies (voir le passage de 2 à 3 variables ci-dessus).

Par exemple, pour la fonction suivante, issue d'une table de vérité comme vu précédemment :

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

CD \ AB	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	1	1	0	1
10	1	1	0	1

Comme dans la table de vérité, chaque terme de la somme se traduit par un 1 dans le tableau. Par exemple, le 1 en haut à droite correspond au terme :

$$A.\overline{B}.\overline{C}.\overline{D}$$

Règles de regroupement

Une fois le tableau de Karnaugh rempli, on cherche à effectuer des regroupements. Les règles de regroupement à respecter sont :

- si on choisit de regrouper les 1, on obtient f, si on choisit les 0, on obtient $\overline{f}$;
- les regroupements doivent porter sur des 1 (resp. des 0) adjacents ;
- les regroupements peuvent être uniquement carrés ou rectangulaires ;
- les regroupements peuvent porter sur 2^n colonnes ou lignes, avec n entier naturel : 2, 4, 8, etc.

Dans l'exemple précédent, on peut rassembler les 1 adjacents par 2 regroupements :

AB / CD	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	1	1	0	1
10	1	1	0	1

Ayant dans cet exemple 2 regroupements, la fonction peut être réduite en une somme de 2 termes. Un regroupement de 2 cases sur une ligne ou une colonne élimine 1 variable ; un regroupement de 2^n cases élimine n variables.

Pour le terme correspondant au regroupement de gauche, le 1 de la sortie est indépendant de la valeur de B et de D, on peut donc le simplifier en :

$$\overline{A}.C$$

De même pour le regroupement de droite :

$$A.\overline{B}$$

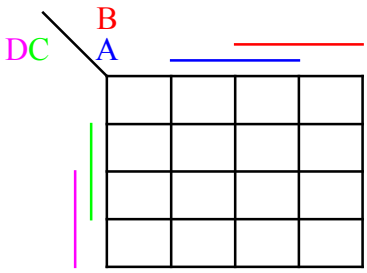
La fonction peut donc se simplifier en :

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

Si les entrées sont composées de 4 variables, on peut les séparer en 2+2 (4 lignes – 4 colonnes). Mais on aurait pu également les séparer en 3+1 (8 lignes – 2 colonnes).

Remarques

- La simplification peut ne pas être unique. Mais les différentes fonctions obtenues sont équivalentes, c'est à dire qu'elles possèdent même table de vérité.
- On peut trouver d'autres présentations du tableau de Karnaugh :



Pour 1 variable :
- présence de la barre = "1"
- absence de la barre = "0"

ou encore

J_1 =	$\overline{A}\overline{B}$	$A\overline{B}$	A.B	$\overline{A}$.B
$\overline{C}\overline{D}$				
C.D				
$\overline{C}$.D				

Exemple pour cinq variables

On cherche à simplifier la fonction suivante.

$$f(A,B,C,D,E)= \Sigma(0, 4, 8, 12, 13, 15, 16, 17, 19, 23, 29, 31)$$

CDE / AB	000	001	011	010	110	111	101	100
00	1							1
01	1					1	1	1
11						1	1	
10	1	1	1			1		

0, 8, 4, 12

Il y a deux groupes de quatre 1 : $\overline{A}\overline{D}\overline{E}$ correspondant aux entrées 0, 4, 8 et 12, et B.C.E à 13, 15, 31, et 29.

Il y a 3 groupes de deux 1 : $A\overline{B}\overline{C}\overline{D}$ correspondant aux entrées 16 et 17, $A\overline{B}\overline{C}$.E à 19 et 23, et A.C.D.E à 23 et 31.

La fonction simplifiée est donc :

$$f(A, B, C, D, E) = \overline{A}\overline{D}\overline{E} + B.C.E + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}.E + ACDE$$

Combinaisons d'entrée non-utilisées

Il peut arriver qu'il y ait des combinaisons non-utilisées dans les variables d'entrée. Ces combinaisons correspondent à une sortie indéterminée. Par exemple, dans la table de vérité suivante :

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

Tous les états possibles d'entrée ne sont pas utilisés. On met une croix dans l'emplacement correspondant du tableau de Karnaugh :

c \ ab	00	01	11	10
0	0	1	x	1
1	1	0	x	x

Si l'on a choisi de regrouper les 1, on remplace ces croix par des 1 de manière à permettre des regroupements plus grands. D'où la fonction simplifiée :

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

La possibilité de simplifier ou non dépend de l'application : si l'on est sûr que les autres combinaisons d'entrée n'auront jamais lieu, on peut considérer ces états comme indifférents et donc simplifier la fonction, comme ci-dessus. Si ces combinaisons ne figurent pas dans la table de vérité mais peuvent physiquement exister, il faut préciser quand même la sortie correspondante (a priori, 0). Dans ce cas, on ne peut pas simplifier plus la fonction.

I.4) Circuits logiques

On a vu que chaque fonction logique élémentaire pouvait être représentée par une porte logique. C'est la représentation qui permet de passer à la réalisation pratique des fonctions logiques, c'est à dire avec des circuits intégrés.

Nous allons voir maintenant comment un circuit logique quelconque peut être réalisée à partir des opérateurs logiques élémentaires simples (NON, OU, ET) et des opérateurs logiques élémentaires composés (NON-OU, NON-ET, OU EXCLUSIF, NON-OU EXCLUSIF)..

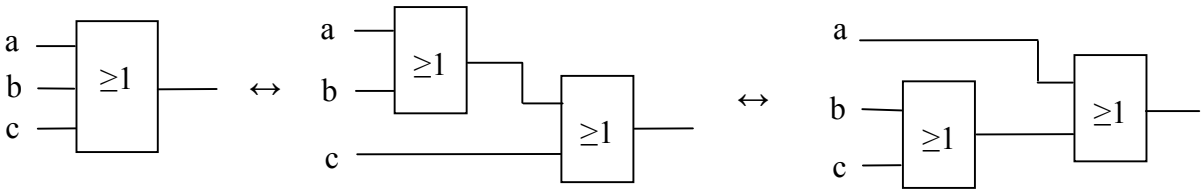
Utilisation de portes à 2 entrées

Il existe des circuits intégrés comportant des portes à nombre d'entrées supérieur à 2. Cependant, on utilise souvent des portes à 2 entrées (notamment pour éviter d'utiliser plusieurs circuits intégrés dans un même montage).

Par exemple, on a vu précédemment que le OU était associatif :

$$F=A+B+C=(A+B)+C=A+(B+C)$$

Cette propriété se traduit au niveau du circuit par les équivalences :



L'inconvénient pratique que comporte cette solution est que physiquement, une porte logique possède un temps de propagation, c'est à dire que lorsque des entrées lui sont appliquées, il existe un petit délai avant que la sortie correspondante ne soit disponible. Donc,

si l'on utilise plusieurs portes en cascade, il faut ajouter les temps de propagation de toutes les portes.

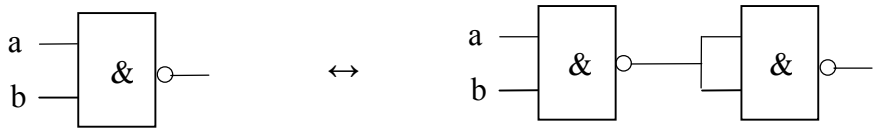
Pour obtenir une fonction quelconque, on associe des portes simples ou composées entre elles, en essayant de réduire leur nombre au maximum, c'est à dire en simplifiant la fonction au préalable.

Portes universelles

Les portes NON-ET et NON-OU sont appelées portes universelles car elles permettent d'obtenir n'importe laquelle des 3 fonctions logiques de base (NON, OU, ET).

Le NON peut être obtenu par une porte NON-ET dont les entrées sont reliées entre elles :

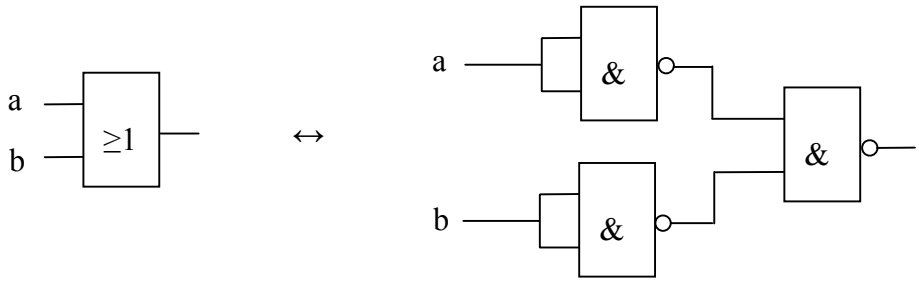
Le ET peut être obtenu par une porte NON-ET en série avec une autre montée en inverseur (=fonction NON) :



Le OU peut être obtenu par 2 NON-ET montés en inverseurs en entrée d'un 3^{e} NON-ET :

Cette équivalence correspond à l'égalité :

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

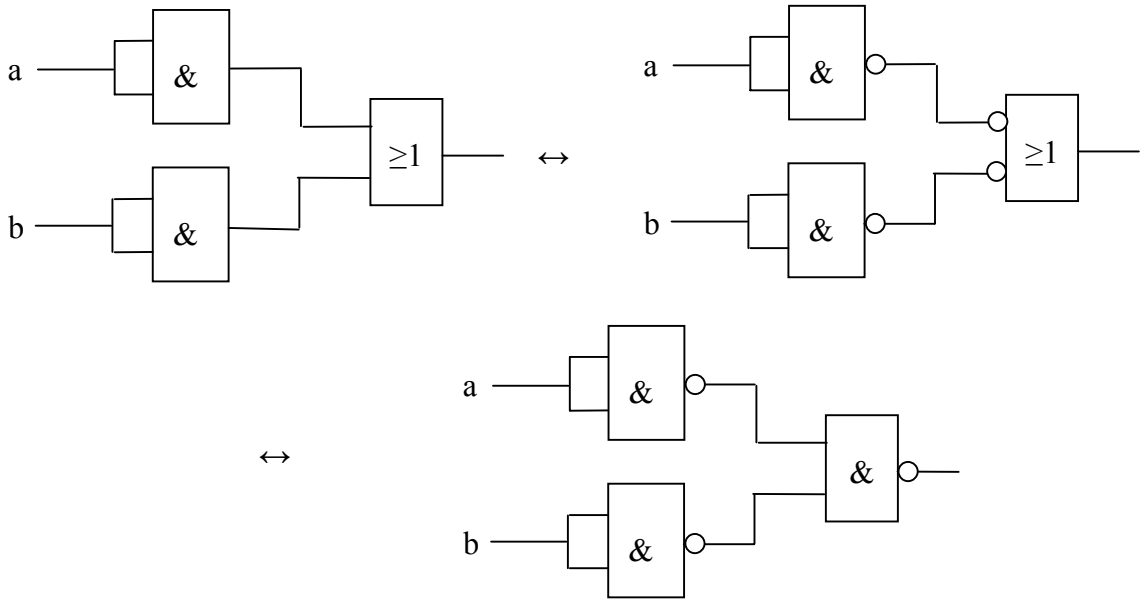


Pour le NON-OU, il suffit de rajouter un inverseur supplémentaire.

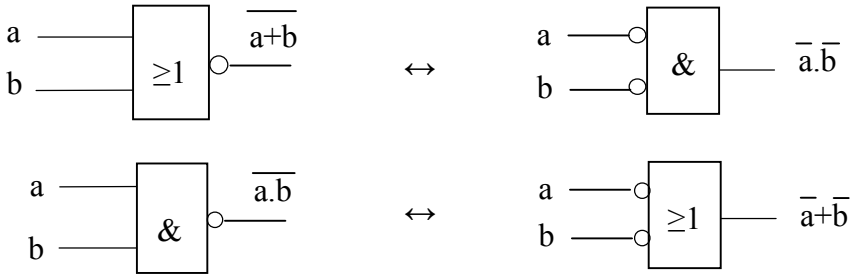
En remplaçant les portes NON-ET par des portes NON-OU, on obtiendrait pour le dernier schéma la fonction ET.

Transformation d'un circuit en portes NON-ET

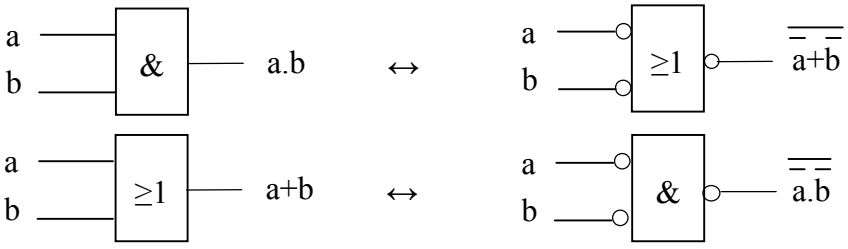
Dans l'exemple suivant, on peut inverser les signaux en sortie de portes, mais également en entrée ; cela peut faciliter les conversions.



La dernière transformation utilise le théorème de DeMorgan. La propriété duale de DeMorgan correspond aux équivalences :



De même :



Partie II) Logique combinatoire

Un circuit combinatoire possède un certain nombre d'entrées et un certain nombre de sorties. Les sorties sont reliées aux entrées par des fonctions logiques. L'aspect temporel n'intervient pas, contrairement aux circuits logiques séquentiels.

Ces circuits sont établis à partir d'une opération appelée synthèse combinatoire. La synthèse combinatoire est la traduction d'une fonction logique, à partir d'un cahier des charges, en un schéma. Diverses méthodes de synthèse sont possibles ; elles diffèrent sur la forme de la fonction utilisée (canonique ou simplifiée), sur le type des opérateurs ou des circuits intégrés choisis, et sur la technique de découpage fonctionnel employée.

Dans cette partie, nous allons étudier quelques grandes fonctions combinatoires couramment utilisées.

II.1) Codeur/décodeur binaire

Ce mot désigne l'ensemble des codeurs, décodeurs et convertisseurs de code. Ces circuits transforment une information codée sous une certaine forme, en une information équivalente mais codée sous une autre forme.

II.1.1) Codeur

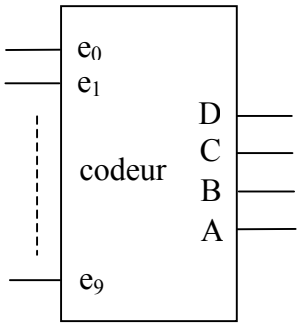
Un codeur (sous-entendu binaire) est un circuit logique codant en sortie l'indice de son entrée active (= "à 1", en logique positive). Sa condition de fonctionnement est donc qu'il n'y ait qu'une seule entrée active en même temps.

Pour n sorties, il peut posséder 2^n entrées.

Un exemple d'application de ce type de codeur est la commande d'une opération d'une machine par un groupe de boutons-poussoirs, dont un seul peut être activé en même temps.

La table de vérité de ce codeur est la suivante (elle comporte une colonne supplémentaire indiquant la valeur décimale N correspondant à l'indice de l'entrée active) :

N	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	A	B	C	D
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
2	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	0	0	0	0	1	0	1
6	0	0	0	1	0	0	0	0	0	0	0	1	1	0
7	0	0	1	0	0	0	0	0	0	0	0	1	1	1
8	0	1	0	0	0	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	1



Par exemple si l'entrée e_6 est active (N=6), le mot de sortie est (D,C,B,A) = (0,1,1,0).

On cherche maintenant à déterminer les fonctions logiques de chacune des sorties, ce qui est nécessaire si l'on veut réaliser ce circuit physiquement.

On pourrait passer par les tableaux de Karnaugh, mais en fait on va voir que cela n'est pas nécessaire. En effet, une seule des entrées étant à 1 en même temps, toutes les combinaisons binaires d'entrée ne sont pas utilisées. Vérifions sur un cas simple à 2 entrées :

N	e_3	e_2	e_1	e_0	B	A
0	0	0	0	1	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	0
3	1	0	0	0	1	1

La fonction logique donnant les valeurs de A correctes sont :

$$A = \overline{e_3}\overline{e_2}.e_1\overline{e_0} + e_3\overline{e_2}\overline{e_1}\overline{e_0}$$

En supposant qu'on n'aura jamais le cas où 2 entrées sont à 1 en même temps, on peut simplifier le tableau de Karnaugh correspondant en le complétant par des états indéterminés:

e_1e_0 \ e_3e_2	00	01	11	10
00		0	x	1
01	0		x	x
11	x	x	x	x
10	1	x	x	x

ce qui donne pour A :

$$A = e_1 + e_3$$

On voit bien quelle est la simplification apportée par le tableau de Karnaugh : la sortie A est la somme des 2 entrées actives pour lesquelles elle vaut 1.

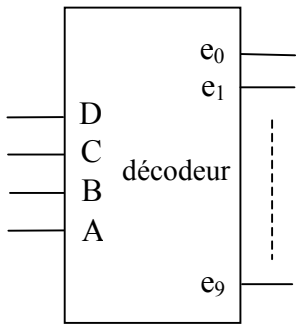
On peut donc déterminer directement la fonction à partir de la table de vérité, et ceci quel que soit le nombre d'entrées.

Ainsi, dans le cas de 10 entrées, on peut déterminer directement :

$$A = e_1 + e_3 + e_5 + e_7 + e_9$$
$$B = e_2 + e_3 + e_6 + e_7$$
$$C = e_4 + e_5 + e_6 + e_7$$
$$D = e_8 + e_9$$

II.1.2) Décodeur

Le décodeur binaire réalise la fonction inverse de celle du codeur. Il possède n entrées et 2^n sorties. On peut considérer que ce circuit code en décimal (chacune des sorties étant associée à un chiffre décimal différent) l'entrée codée en binaire.



Par exemple, quand on a (D,C,B,A) = (0,0,1,1), la sortie d'indice 3 (e_3) est à 1.

Structure logique interne

Elle se déduit de la table de vérité :

A	B	C	D	N	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	2	0	0	0	0	0	0	0	1	0	0
0	0	1	1	3	0	0	0	0	0	0	1	0	0	0
0	1	0	0	4	0	0	0	0	0	1	0	0	0	0
0	1	0	1	5	0	0	0	0	1	0	0	0	0	0
0	1	1	0	6	0	0	0	1	0	0	0	0	0	0
0	1	1	1	7	0	0	1	0	0	0	0	0	0	0
1	0	0	0	8	0	1	0	0	0	0	0	0	0	0
1	0	0	1	9	1	0	0	0	0	0	0	0	0	0

N est la valeur décimale correspondante à l'indice de la sortie active.
On obtient les fonctions suivantes s_0, …s_9 directement sous forme somme-de-produits :

$$s_0 = \overline{A}.\overline{B}.\overline{C}.\overline{D}, \quad s_1 = \overline{A}.\overline{B}.\overline{C}.D, \quad s_2 = \overline{A}.\overline{B}.C.\overline{D}, \quad \text{etc}$$

Réalisation d'une fonction logique quelconque au moyen d'un décodeur

Les décodeurs permettent de réaliser n'importe quelle fonction logique. Précédemment, on avait défini les fonctions logiques directement à partir des variables d'entrée. Le résultat correspondait à une réalisation de la fonction en portes logiques élémentaires. L'utilisation d'un décodeur constitue donc une autre manière de réaliser une fonction.

Le problème est similaire à la détermination de la fonction logique d'un codeur vue dans le paragraphe précédent. On définit d'abord la fonction à réaliser en fonction des sorties s_i du décodeur, de la même manière qu'on avait déterminé les sorties du codeur.

Exemple

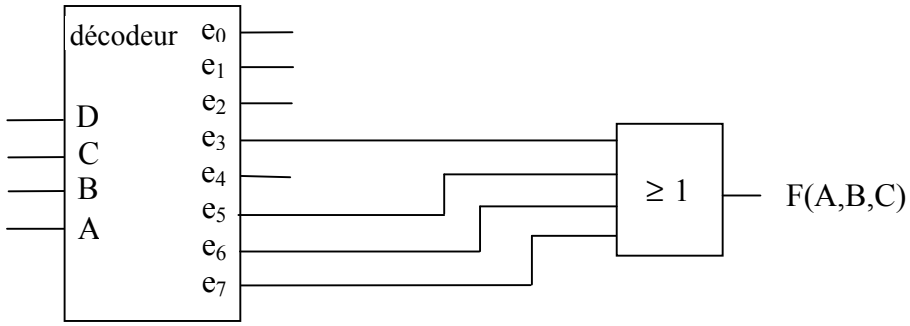
On cherche à réaliser la fonction f (A, B, C) = Σ (3, 5, 6, 7) avec un décodeur et une porte logique. On remplit la table de vérité de manière adéquate :

A	B	C	N	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	F(A,B,C)
0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	1	0	0
0	1	0	2	0	0	0	0	0	1	0	0	0
0	1	1	3	0	0	0	0	1	0	0	0	1
1	0	0	4	0	0	0	1	0	0	0	0	0
1	0	1	5	0	0	1	0	0	0	0	0	1
1	1	0	6	0	1	0	0	0	0	0	0	1
1	1	1	7	1	0	0	0	0	0	0	0	1

On en déduit alors l'expression de f, de la même manière qu'on l'avait fait pour le codeur :

$$F(A,B,C) = s_3 + s_5 + s_6 + s_7$$

On peut alors en déduire le schéma correspondant :



II.2) Transcodeurs

Un transcodeur est un convertisseur de code. Il existe un grand nombre de transcodeurs possibles. Ce paragraphe décrit un type de transcodeur couramment utilisé : le transcodeur 7 segments.

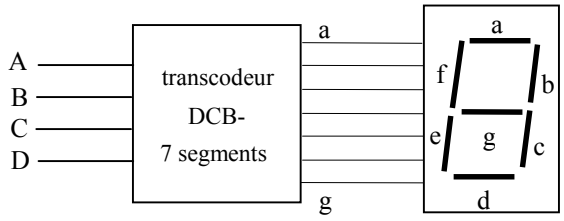
II.2.1) Transcodeur DCB-7 segments

Un transcodeur DCB-7 segments permet de générer les signaux logiques adéquats pour afficher sur un afficheur à 7 segments la valeur décimale (de 0 à 9) correspondant à la valeur binaire de l'entrée.

Un afficheur à 7 segments est composé de 7 LEDs (pour Light Emitting Diode, ou DEL pour Diode Emettrice de Lumière, en français), pouvant être commandées par des niveaux logiques. Selon le type de LED utilisé, un niveau 1 peut se traduire par l'allumage ou son extinction, et inversement pour le niveau 0.

Dans les applications pratiques (par exemple, affichage de la somme d'argent entrée dans un distributeur de boissons), on veut afficher les valeurs décimales (de 0 à 9) et non pas les valeurs hexadécimales (de 0 à 9, puis A à F). C'est pourquoi on parle de Décimal Codé en Binaire (BCD en anglais). Cela signifie que l'on n'utilise que les 10 premières valeurs du code hexadécimal qui en comporte 16 (codés sur 4 bits).

Sur le schéma ci-dessous, les 4 entrées sont appelées A, B, C, D et les 7 sorties a, b, c, d, e, f, g.



Chacune des 7 sorties de ce transcodeur correspond à une fonction logique différente. La table de vérité correspondante comporte donc 4 entrées et 7 sorties.

Supposons que les afficheurs utilisés nécessitent un 1 logique pour l'allumage de ses segments. Pour chaque ligne de la table de vérité, on place des 1 dans une colonne lorsque l'on souhaite que le segment correspondant soit allumé pour l'entrée codée dans cette ligne.

Par exemple, pour la combinaison

$$(A, B, C, D) = (0, 0, 1, 0)$$

on souhaite que le chiffre 2 s'affiche. Pour cette ligne, les colonnes correspondant aux segments a, b, d, e, g doivent donc comporter un 1.

Une fois la table de vérité déterminée, il reste à simplifier les fonctions des 7 segments. On peut utiliser pour cela des tableaux de Karnaugh.

Il est important de remarquer toutes les combinaisons possibles ne sont pas utilisées. Comme indiqué plus haut, les "trous" correspondant dans les tableaux de Karnaugh correspondent aux combinaisons d'entrée non-utilisées. Si l'on est sûr que ces combinaisons n'apparaîtront jamais sur les entrées, on peut considérer les états correspondants des sorties comme indéterminés. On les remplace par des 0 ou des 1 selon les cas, pour simplifier au maximum les fonctions. Par contre, si ces combinaisons peuvent se produire et qu'on souhaite qu'elles ne provoquent pas d'allumage des LEDs, il faut mettre les sorties correspondantes à 0.

II.2.2) Autres transcodeurs

Le transcodage est un principe général dans lequel le nombre de bits du code d'entrée et celui du code de sortie peuvent être quelconques. On peut par exemple vouloir réaliser un transcodage du code binaire naturel vers le code de Gray, ou bien l'inverse.

Le transcodeur DCB-7 segments étudié au paragraphe précédent n'est qu'un exemple d'application pratique de ce principe. Le codeur et le décodeur sont deux cas particuliers de transcodeurs.

Quel que soit le transcodeur recherché, la méthode de synthèse est exactement la même que celle utilisée pour le transcodeur DCB-7 segments. De plus, comme pour ce dernier, il peut y avoir des combinaisons d'entrées non-utilisées, auxquelles correspondent des états indéterminés des sorties. Ces derniers peuvent être mis à profit pour simplifier les équations logiques des sorties.

II.3) Multiplexeur/démultiplexeur

Les multiplexeurs et démultiplexeurs sont des circuits combinatoires couramment utilisés. Les premiers permettent, entre autre, de transformer des données parallèles en

données série, d'utiliser un même circuit intégré pour plusieurs signaux (et donc de simplifier les circuits), mais également de synthétiser n'importe quelle fonction logique. Les seconds permettent, entre autres, de réaliser la conversion série-parallèle.

II.3.1) Multiplexeur

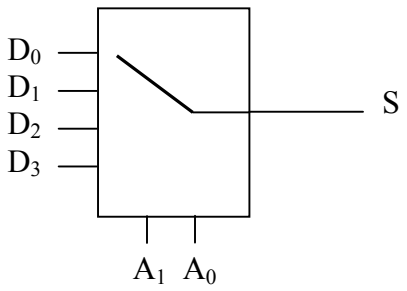
Principe

C'est un circuit qui possède

- des entrées de commande (ou d'adresse), au nombre de n : $A_{n-1}…A_0$;
- des entrées de données, au nombre de 2^n : $D_{2^n-1}…D_0$;
- une sortie S.

Un multiplexeur est qualifié par rapport à ses entrées de données. Par exemple, on parle de multiplexeur 8 vers 1 pour 8 entrées de donnée et 3 entrées d'adresse. Le multiplexeur peut être vu comme un commutateur commandé par les entrées d'adresse : selon la valeur des entrées d'adresse, une des entrées de données sera connectée en sortie.

Supposons un multiplexeur à deux entrées adresses A_1 et A_0 ; pour chaque combinaison binaire de (A_1, A_0), une entrée de donnée sera connectée en sortie. Par exemple, pour (A_1, A_0)=10, l'entrée D_2 sera connectée en sortie.



La table de vérité correspondante est la suivante :

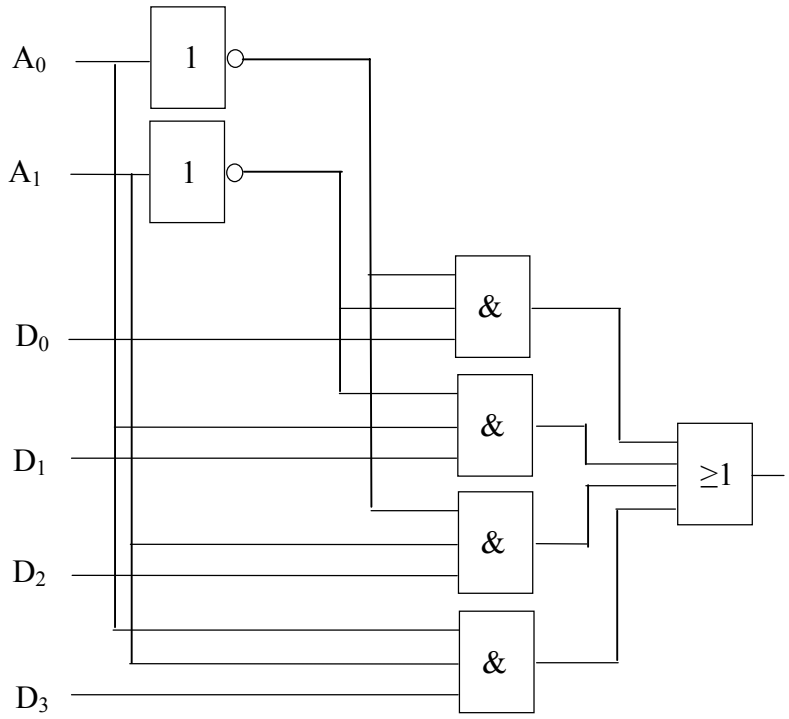
A_1	A_0	Entrée sélectionnée
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

On en déduit la fonction logique de sortie. Chaque ligne de la table correspond à un terme d'une somme de produits. On détermine chacun de ces termes de la même façon que pour une fonction logique composée de 0 et de 1, sauf que chaque combinaison d'entrée est multipliée par la valeur correspondante de la fonction :

$$S = D_0.\overline{A_0}.\overline{A_1} + D_1.A_0.\overline{A_1} + D_2.\overline{A_0}.A_1 + D_3.A_0A_1$$

Réalisation

Le schéma se déduit directement de l'expression de la sortie :

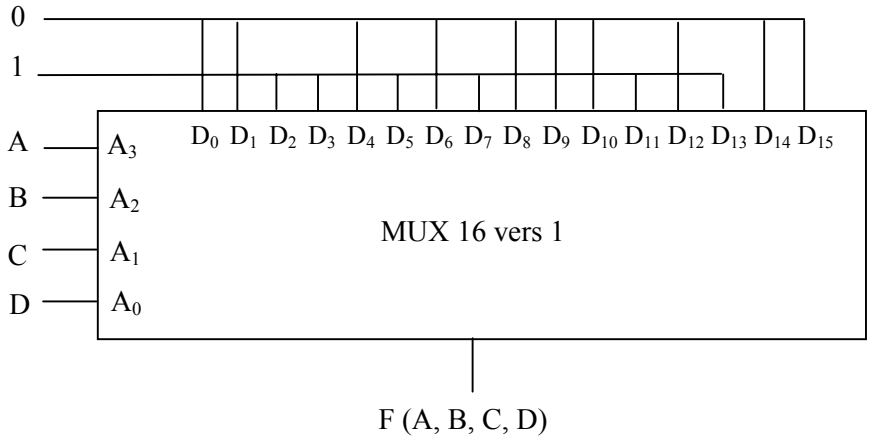


Exemple d'application d'un multiplexeur : réalisation d'une fonction quelconque

Soit la fonction $F(A, B, C, D) = \sum (2, 3, 5, 7, 11, 13)$.

On cherche à réaliser cette fonction avec une multiplexeur à quatre entrées d'adresses, A_0 à A_3, et 2^4=16 entrées de données, D_0 à D_{15}.

En utilisant les variables A, B, C, D de F comme entrées d'adresse du multiplexeur, il suffit de relier l'entrée sélectionnée par chaque adresse à 0 ou à 1, selon que F vaut 0 ou 1 pour la combinaison correspondante des variables. On obtient donc le schéma suivant :



On peut également réaliser cette même fonction avec un multiplexeur possédant moins d'entrées d'adresse que de variables. Il faut alors exprimer la fonction en fonction des variables non-utilisées comme entrées d'adresse.

Pour cela on peut s'aider de la table de vérité de F :

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Par exemple, avec 3 entrées d'adresse seulement, une solution consiste à utiliser 3 des variables comme entrées d'adresse (par exemple A, B et C), et d'exprimer F en fonction de la variable non utilisée (ici, D).

Chacune des combinaisons des entrées d'adresse occupe 2 lignes. Sur ces 2 lignes, il faut exprimer F en fonction de D. On aura 4 cas possibles : F=0, F=1, F=D ou F=$\overline{D}$.

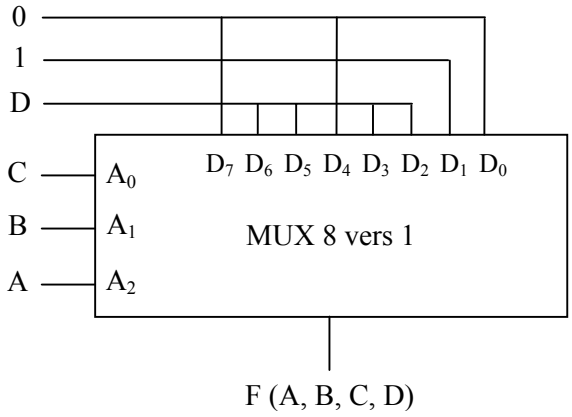
Par exemple, la première adresse : (A, B, C)=(0, 0, 0) correspond aux 2 première lignes de la table de vérité. Il faut trouver l'expression de F sur ces 2 lignes. On constate que l'on a F=0. Il faut donc relier l'entrée D_0 du multiplexeur (celle qui est sélectionnée par l'adresse (0,0,0)) à 0 :

ces 2 lignes correspondent à l'adresse (0,0,0) du multiplexeur

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1

sur ces 2 lignes, F=0

On applique le même principe pour toutes les autres combinaisons de (A,B,C). On obtient le schéma suivant :



Remarque : on aurait pu utiliser n'importe laquelle des 4 variables pour les 3 entrées d'adresse du multiplexeur. La lecture de l'expression de F à partir de la table de vérité aurait simplement été un peu moins directe (il aurait fallu regarder la valeur de F sur 2 lignes non-adjacentes).

Avec une multiplexeur à deux entrées d'adresse seulement., le principe à appliquer est le même que précédemment : il faut exprimer F pour chacune des adresses, en fonction des variables non-utilisées comme entrées d'adresse.

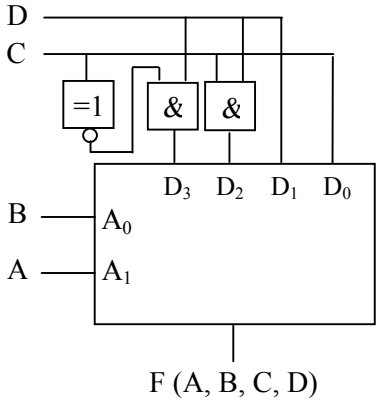
On choisit d'utiliser les variables A et B comme entrées d'adresse. Par exemple, pour la première adresse (0,0), on détermine l'expression de F en fonction de C et D (les 2 variables non-utilisées comme entrées d'adresse), sur les 4 premières lignes de la table de vérité :

ces 4 lignes correspondent à l'adresse (0,0) du multiplexeur

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1

sur ces 4 lignes, on constate que F=D

et ainsi de suite. On obtient alors le schéma complet suivant :

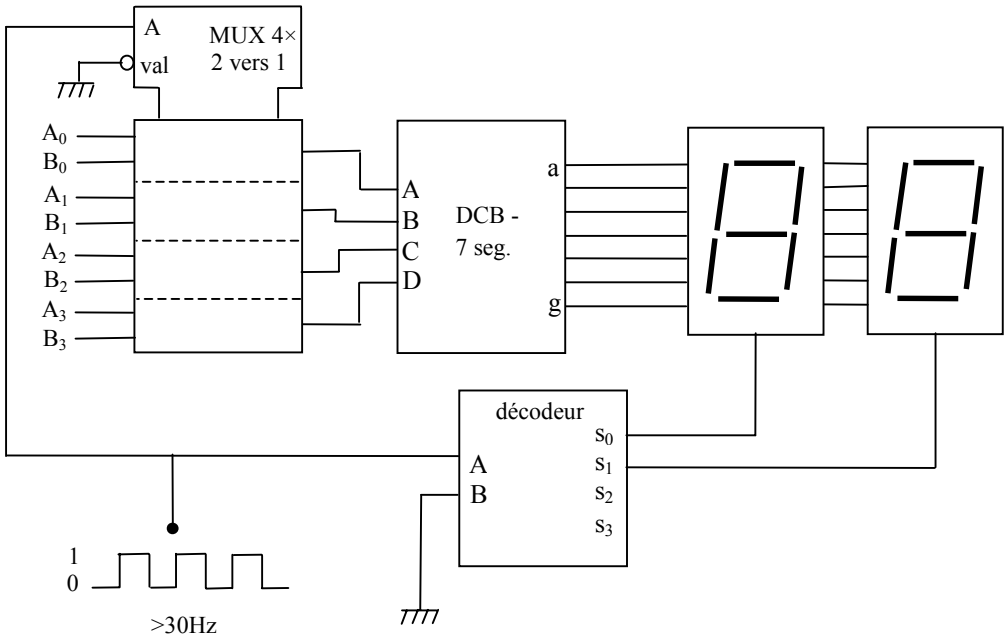


Autre exemple d'application : commande de plusieurs afficheurs 7 segments

Un multiplexeur permet également d'utiliser un même circuit intégré pour traiter plusieurs données. Par exemple, on peut utiliser un seul transcodeur BCD-7 segments pour commander plusieurs afficheurs à 7 segments.

Par exemple, on souhaite afficher 2 chiffres décimaux sur 2 afficheurs à 7 segments. Ces 2 chiffres sont codés en binaire, sur 2 mots de 4 bits : A_3, A_2, A_1, A_0 et B_3, B_2, B_1, B_0. L'idée est d'utiliser un multiplexeur pour présenter alternativement ces 2 mots binaires en entrée du transcodeur BCD-7 segments. Si, dans le même temps et de manière synchrone, on active alternativement les 2 afficheurs 7 segments, et si la fréquence de cette commutation est supérieure à 20-30Hz, l'utilisateur verra chacun des 2 chiffres sur un afficheur dédié. Le multiplexeur à utiliser ici est un quadruple 2-vers-1. L'activation d'un seul des 2 afficheurs peut être obtenue au moyen d'un codeur, commandé par le même signal que celui commandant le multiplexeur (sur son unique entrée d'adresse A).

Un schéma implantant cette idée pourrait être le suivant :



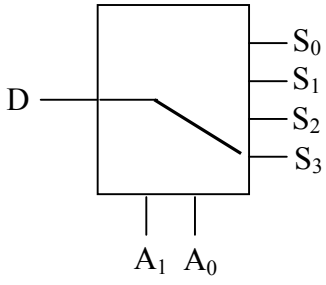
II.3.2) Démultiplexeurs

Un démultiplexeur réalise l'opération inverse de celle du multiplexeur. Il s'agit d'un circuit possédant :

- n entrées de commande (ou d'adresse) : $A_{n-1}…A_0$;
- 2^n sorties : $S_{2^n-1}…S_0$;
- une entrée de donnée D.

Un démultiplexeur recopie l'entrée D sur la sortie correspondant à la valeur présente sur les entrées d'adresse. Une sortie non sélectionnée est à 0.

Pour un démultiplexeur à 2 entrées d'adresse, si on a par exemple (A_1, A_0)= 11, la sortie 3 est connecté à D :



La table de vérité correspondante est la suivante :

A_1	A_0	S_3	S_2	S_1	S_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Les expressions logiques des sorties s'en déduisent :

$$S_0 = D.\overline{A_1}.\overline{A_0}\,, \quad S_1 = D.\overline{A_1}.A_0\,, \quad S_2 = D.A_1.\overline{A_0}\,, \quad S_3 = D.A_1.A_0$$

On peut remarquer que lorsque l'entrée vaut toujours 1, un démultiplexeur devient un décodeur (et donc qu'un décodeur est un cas particulier du multiplexeur).

II.4) Comparateur

Un comparateur permet de comparer 2 mots binaires, c'est à dire d'indiquer si ces 2 mots sont égaux, mais également, si ça n'est pas le cas, lequel est le plus grand. Il est basé sur l'utilisation d'un comparateur 1 bit.

II.4.1) Comparateur d'égalité

Deux mots binaires sont égaux si leurs éléments binaires de même poids sont égaux deux à deux. Par exemple, 2 mots de 4 bits A (A_3, A_2, A_1, A_0) et B (B_3, B_2, B_1, B_0) sont égaux si $A_3 = B_3$ et $A_2 = B_2$ et $A_1 = B_1$ et $A_0 = B_0$.

On a vu plus haut que la fonction NON OU-EXCLUSIF à 2 entrées était un détecteur d'égalité entre ces 2 bits, c'est à dire que la fonction valait 1 si ces 2 bits étaient égaux, 0 s'ils étaient différents.

Pour comparer plusieurs bits, il suffit donc d'associer plusieurs fonctions NON OU-EXCLUSIF en série. La fonction permettant de détecter l'égalité des 2 mots binaires de l'exemple est donc :

$$f(A = B) = \left(\overline{A_3 \oplus B_3}\right)\left(\overline{A_2 \oplus B_2}\right)\left(\overline{A_1 \oplus B_1}\right)\left(\overline{A_0 \oplus B_0}\right)$$

II.4.2) Comparateur complet

On souhaite réaliser un comparateur complet de deux mots de 4 bits. Un comparateur complet de 1 bit possède non seulement une sortie signalant l'identité de ses 2 bits d'entrée, mais également une sortie signalant que la 1ère est supérieure à la 2e, et une sortie signalant l'inverse. Soient a_0 et b_0 ces 2 entrées. Appelons E, S et I respectivement ces 3 sorties :

- E=1 si $a_0 = b_0$,
- S=1 si $a_0 > b_0$,
- I=1 si $a_0 < b_0$.

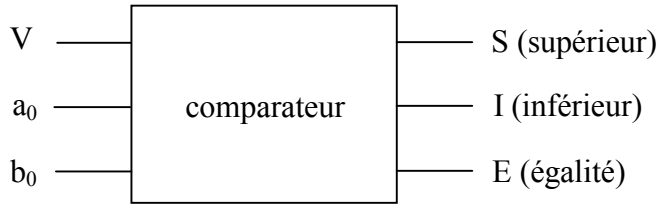
La table de vérité de ce comparateur est la suivante :

a_0	b_0	S	I	E
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

On en déduit les équations des sorties :

$$S = a_0\overline{b_0}\ ,\ I = \overline{a_0}b_0\ ,\ E = \overline{a_0 \oplus b_0}$$

Le comparateur peut être schématisé par le bloc fonctionnel suivant. L'entrée V est une entrée de validation. Le comparateur fonctionne si V est égal à 1, sinon toutes les sorties sont égales à 0 (on en verra l'utilité dans l'association de plusieurs de ces blocs).



Nous cherchons maintenant à réaliser un comparateur de deux mots de quatre bits A(a_0, a_1, a_2, a_3) et B(b_0, b_1, b_2, b_3) à l'aide de comparateurs 1-bit mis en cascade et de quelques portes logiques.

Le comparateur doit indiquer s'il y a égalité de ses 2 mots binaires d'entrée, et sinon lequel est le plus grand, par l'intermédiaire de ses 3 sorties E, S et I.

L'idée est de comparer les bits correspondants 1 à 1 en partant de celui de poids le plus fort.

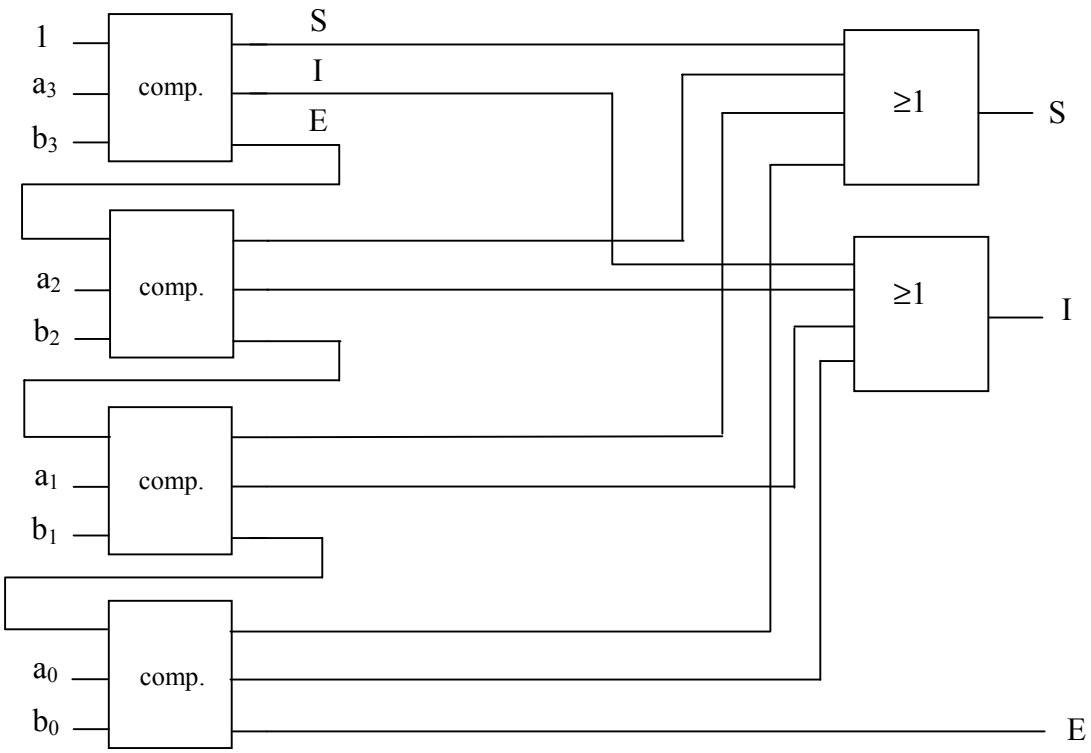
Si le bit de poids le plus fort du 1er mot est 1 et celui du 2^{e} sont les mêmes, on n'est pas renseigné sur la comparaison de ces 2 mots donc il faut aller voir le bit de poids immédiatement inférieur. Et ainsi de suite jusqu'aux bits de poids le plus faible.

Si l'un des bits est 0 et le bit correspondant de l'autre mot est 1, il n'est pas nécessaire d'aller tester les bits de poids plus faibles, on connaît le résultat. Il faut alors utiliser les informations de S et de I au moyen d'une porte OU. Pour réaliser ceci, on peut relier la sortie locale E_i d'un comparateur 1-bit à l'entrée de validation du suivant ; ainsi, dès qu'un bit sera différent du bit correspondant de l'autre mot, E sera égale à 0 et inhibera la comparaison des bits de poids inférieurs.

Si les 2 mots sont égaux, la sortie E du comparateur des bits de poids les plus faibles sera à 1. Elle peut donc constituer la sortie d'égalité des 2 mots complets.

La sortie S (supérieur) doit être à 1 si l'une des sorties S_i est à 1. Elles doivent donc être combinées par une porte OU (à 4 entrées ici). C'est le même principe pour la sortie I.

On obtient donc le schéma suivant pour ce comparateur :



Un des inconvénients de cette structure est que le résultat de la comparaison apparaît après un temps lié aux nombres de portes logiques traversées. Pour palier à cet inconvénient, il faudrait utiliser une structure parallèle (non étudiée ici).

II.5) Les Additionneurs

Il s'agit ici de l'addition arithmétique. Le symbole utilisé est le +, mais l'opération est différente de la somme logique.

En effet, avec la somme logique, on a :

$$1+1=1$$

alors qu'avec l'addition arithmétique, on a :

$$1+1=10$$

On va voir que l'addition arithmétique sur 1 bit s'apparente au OU Exclusif.

II.5.1) Demi additionneur

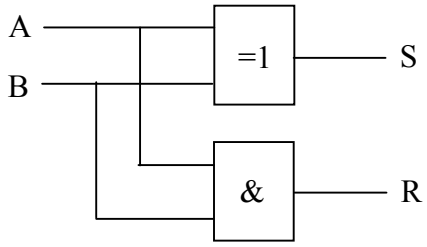
La table de vérité d'un additionneur de deux bits A et B comporte 2 sorties : la sortie S et la retenue R :

A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

On en déduit directement les équations logiques de ces 2 sorties :

$$S = A \oplus B$$
$$R = A.B$$

Le schéma correspondant est donc :



On appelle ce système logique un demi-additionneur, car il ne prend pas en compte une éventuelle retenue provenant du résultat de l'addition des 2 bits de rang directement inférieur.

II.5.2) Additionneur complet

Un additionneur complet comporte 3 entrées : les deux bits à additionner A et B, et la retenue issue de l'addition des 2 bits de rang inférieur (dite entrante), R_{n-1}.

Il possède 2 sorties : la somme S et la retenue sortante R_n.

Sa table de vérité est la suivante :

R_{n-1}	A	B	R_n	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

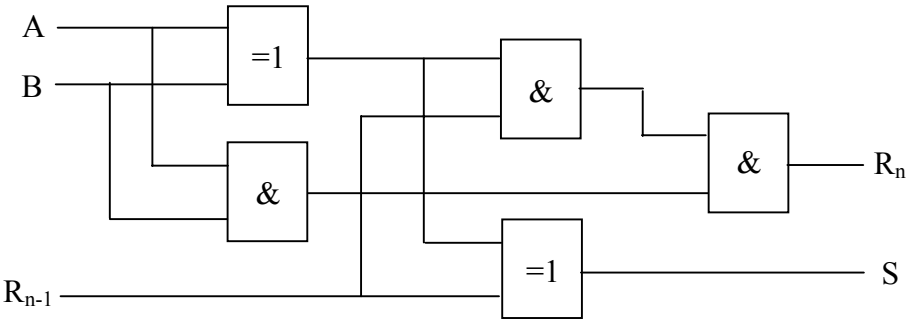
L'expression de la sortie somme S est :

$$S = \overline{R_{n-1}}.\overline{A}.B + \overline{R_{n-1}}.A.\overline{B} + R_{n-1}.\overline{A}.\overline{B} + R_{n-1}.A.B$$
$$= \overline{R_{n-1}}.(A \oplus B) + R_{n-1}.\overline{(A \oplus B)}$$
$$\leftrightarrow \quad S = R_{n-1} \oplus (A \oplus B)$$

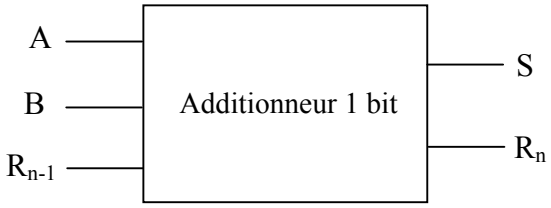
L'expression de la retenue de sortie R_n est :

$$R_n = \overline{R_{n-1}}.A.B + R_{n-1}.\overline{A}.B + R_{n-1}.A.\overline{B} + R_{n-1}A.B$$
$$= R_{n-1}.(\overline{A}.B + A.\overline{B}) + A.B.(\overline{R_{n-1}} + R_{n-1})$$
$$\leftrightarrow \quad R_n = R_{n-1}.(A \oplus B) + A.B$$

Le schéma correspondant est :

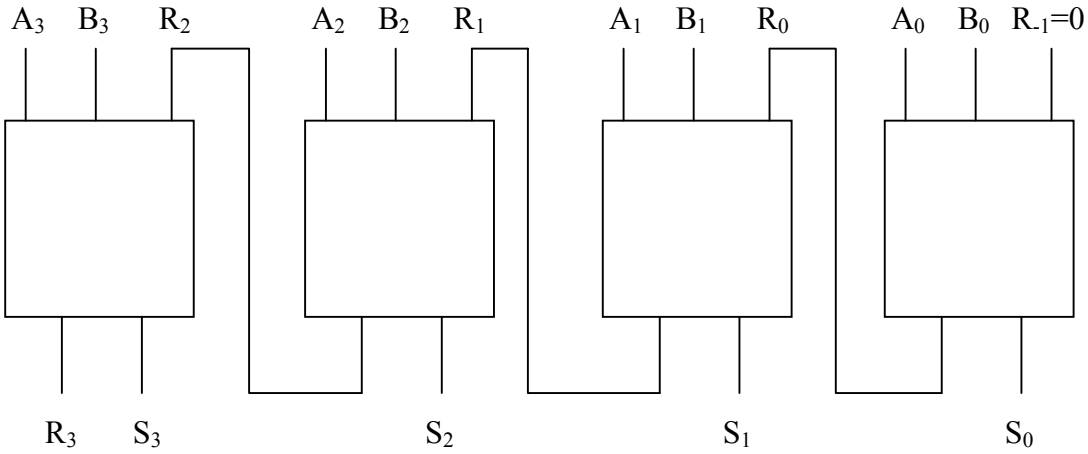


On peut représenter ce circuit sous la forme d'une boite noire :



II.5.3) Additionneur de deux mots à propagation de retenue

L'addition de deux mots de n bits nécessite n additionneurs. La retenue se propage des éléments binaires de poids le plus faible vers les éléments binaires de poids le plus fort. Le schéma suivant présente un additionneur de mots de 4 bits :



Cette architecture est intéressante d'un point de vue matériel car elle est répétitive. Par contre, le résultat obtenu dépend du nombre d'additionneurs donc de la taille des mots à additionner. La retenue R_1 est délivrée après la première addition et ainsi de suite.

II.5.4) Additionneur à anticipation de retenue

Le principal inconvénient de l'additionneur à propagation de retenue est que le temps de calcul est proportionnel au nombre de bits additionnés. En effet, le calcul de l'addition 1 bit à un rang donné nécessite de connaître la retenue obtenue à l'addition du rang directement inférieur.

Pour pallier à cet inconvénient, on peut imaginer de calculer la retenue à un rang donné, à partir de tous les bits d'entrée des rangs inférieurs. C'est ce qu'on appelle l'anticipation de retenue.

Le gain de temps se fait alors au détriment d'une complexité accrue en terme de circuits.

Reprenons l'expression de la retenue au rang n :

$$R_n = R_{n-1}.(A_n \oplus B_n) + A_n.B_n$$

On a :

$$\begin{aligned}
R_0 &= R_{-1}.(A_0 \oplus B_0) + A_0.B_0 \\
&= A_0.B_0 \\
R_1 &= R_0.(A_1 \oplus B_1) + A_1.B_1 \\
&= A_0.B_0.(A_1 \oplus B_1) + A_1.B_1 \\
R_2 &= R_1.(A_2 \oplus B_2) + A_2.B_2 \\
&= (A_0.B_0.(A_1 \oplus B_1) + A_1.B_1).(A_2 \oplus B_2) + A_2.B_2 \\
&= A_0.B_0.(A_1 \oplus B_1).(A_2 \oplus B_2) + A_1.B_1.(A_2 \oplus B_2) + A_2.B_2
\end{aligned}$$

etc.

Pour les sommes, on a :

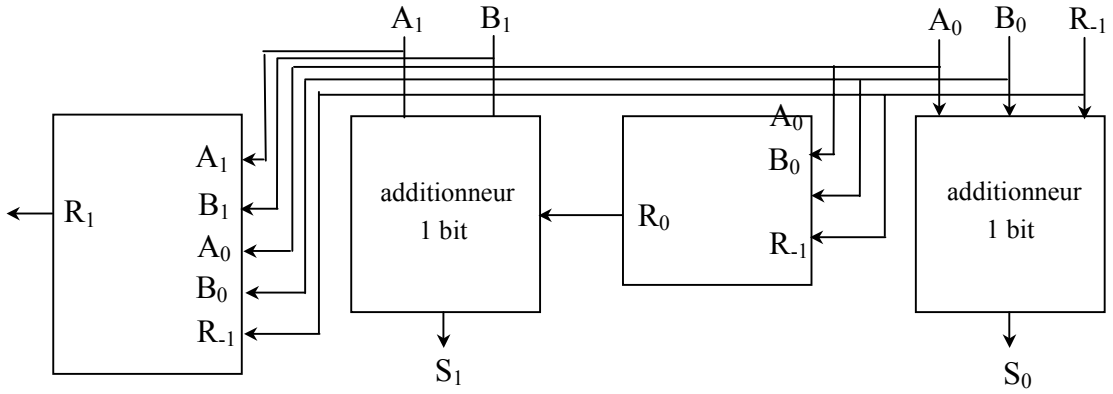
$$S_n = R_{n-1} \oplus (A_n \oplus B_n)$$

soit :

$$\begin{aligned}
S_0 &= R_{-1} \oplus (A_0 \oplus B_0) \\
&= (A_0 \oplus B_0) \\
S_1 &= R_0 \oplus (A_1 \oplus B_1) \\
S_2 &= R_1 \oplus (A_0 \oplus B_0)
\end{aligned}$$

etc.

A chaque étage, on ré-utilise donc toutes les entrées a_i et b_i et la retenue initiale R_{-1} (qui est prise nulle ici). Le schéma correspondant à cet additionneur, pour 2 bits, pourrait donc être le suivant :



Partie III) Logique séquentielle

On distingue deux types de systèmes logiques séquentiels :

- les circuits séquentiels asynchrones, dans lesquels les sorties évoluent dès qu'il y a un changement sur l'une des entrées ;
- les circuits séquentiels synchrones, dans lesquels les sorties ne peuvent évoluer que si un signal d'horloge est actif. Ce dernier peut être actif sur un niveau (0 ou 1) ou sur un front (montant ou descendant).

III.1) Bascules

Les bascules sont les circuits logiques de base de la logique séquentielle. Il existe des bascule asynchrones et des bascules synchrones.

III.1.1) Bascules asynchrones

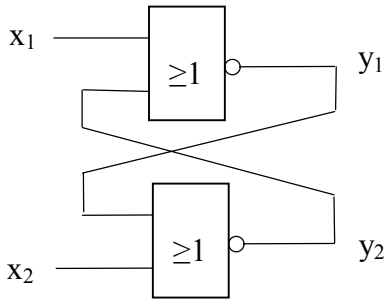
La bascule RS est la plus simple de toutes les bascules. Elle est à la base des autres : la bascule JK et la bascule D, entre autres.

a) Bascule RS

On peut réaliser une bascule RS au moyen de portes NON-OU ou de portes NON-ET.

Bascule RS à portes NON-OU

Etudions le circuit bouclé suivant, à 2 entrées et 2 sorties :



- Le niveau logique 1 étant l'élément absorbant du OU, si $x_1=1$ et $x_2=1$, on a forcément $y_1=y_2=0$.
- Si $x_1=1$ et $x_2=0$, on a $y_1=0$ et donc $y_2=1$.
- Pour les mêmes raisons, si $x_1=0$ et $x_2=1$, on a $y_2=0$ et donc $y_1=1$.
- Si $x_1=0$ et $x_2=0$, ces 2 entrées n'ont plus aucune influence sur les sorties ; ces dernières restent dans l'état dans lequel elles sont. On a en permanence :

$$y_1 = \overline{y_2} \text{ et } y_2 = \overline{y_1}$$

et on peut constater qu'il s'agit d'un état stable.

Ce dernier état correspond à une mémorisation. Il faut alors distinguer l'instant d'application des nouvelles entrées, qui sera indicé n+1, et l'instant précédent, indicé n. Pour formaliser cette mémorisation, on aura donc :

$$y_1(n+1) = y_1(n) \quad \text{et} \quad y_2(n+1) = y_2(n)$$

(avec $y_1 \neq y_2$).

Rassemblons ces résultats dans un tableau :

x_1	x_2	$y_1(n+1)$	$y_2(n+1)$
0	0	$y_1(n)$	$y_2(n)$
0	1	1	0
1	0	0	1
1	1	0	0

Si l'on considère la sortie y_1 comme la sortie principale, les entrées x_1 et x_2 ont des rôles bien distincts. Si on est en logique positive (dans laquelle le niveau actif est le niveau 1, ce qui est en général le cas), l'entrée x_1 provoque la mise à 0 de la sortie y_1 lorsqu'elle est active ; l'entrée x_2 provoque sa mise à 1. On appelle l'entrée x_1 "R" pour RESET, et l'entrée x_2 "S" pour SET.

On remarque également que les 2 sorties sont complémentaires pour les 2^e et 3^e lignes de la table de vérité (et donc pour la 1ère ligne également, si l'on y va à partir de la 2^e ou de la 3^e). On appellera donc la sortie principale Q et l'autre $\overline{Q}$, et on considèrera le 4^e cas comme un cas inutilisé (on dit également interdit).

On peut alors ré-écrire la table de vérité sous la forme suivante (sans indiquer la 2^e sortie, puisqu'on considère qu'il s'agit de la sortie complémentaire à la 1ère) :

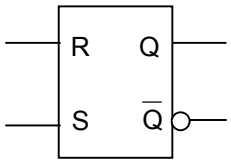
R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	non utilisé (0)

On peut considérer cette bascule comme un élément de mémorisation de 1 bit.

Pour résumer, une bascule RS comporte :

- une entrée R (Reset) de mise à zéro,
- une entrée S (Set) de mise à un,
- une sortie Q et son inverse $\overline{Q}$.

Il s'agit d'un bloc fonctionnel logique à part entière ; on peut le symboliser par une boîte noire représentant ses entrées et ses sorties :



Pour déterminer la fonction Q_{n+1}, il faut considérer Q_n comme une des entrées, puisqu'il s'agit d'une variable pouvant prendre les valeurs 0 ou 1. On peut alors établir le tableau de Karnaugh, comme on l'a fait pour les circuits combinatoires :

Q_n \ RS	00	01	11	10
0	0	1	0	0
1	1	1	0	0

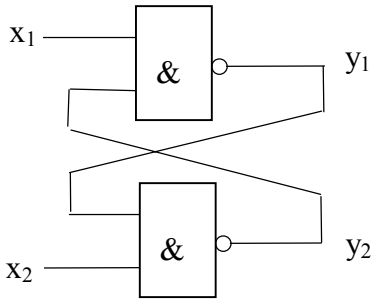
d'où

$$Q_{n+1} = \overline{R}.S + \overline{R}.Q_n$$

$$\leftrightarrow \quad Q_{n+1} = \overline{R}(S + Q_n)$$

Bascule RS à portes NON-ET

Etudions maintenant le même circuit bouclé mais avec des portes NON-ET :



Le raisonnement à tenir est le même que pour la version à portes NON-OU, sauf que l'élément absorbant est ici le 0.

La table de vérité devient :

x_1	x_2	$y_1(n+1)$	$y_2(n+1)$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	$y_1(n)$	$y_2(n)$

Si l'on considère la sortie y_1 comme la sortie principale Q, comme précédemment, la table de vérité devient :

x_1	x_2	Q_{n+1}
0	0	non utilisé (1)
0	1	1
1	0	0
1	1	Q_n

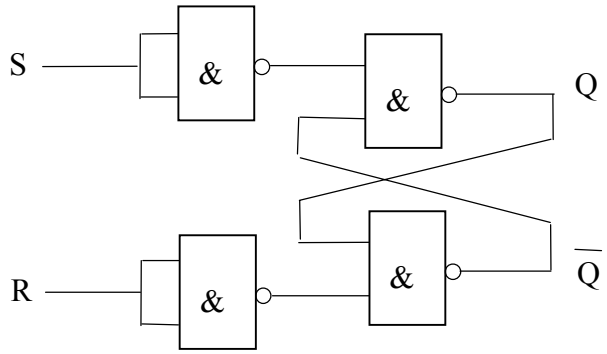
On voit que l'entrée x_1 n'a plus un rôle de mise à 0 mais un rôle de mise à 1 et x_2 un rôle de mise à 0. Pour que cette nouvelle table de vérité corresponde à celle de la bascule RS à portes NOR, il faut donc d'abord complémenter ces 2 entrées, ce qui donne :

$\overline{x_1}$	$\overline{x_2}$	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	1 (non utilisé)

Il reste à considérer que :

$$R = \overline{x_2} \quad \text{et} \quad S = \overline{x_1}$$

Le schéma et la table de vérité correspondante deviennent donc :



R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	1 (non utilisé)

Un raisonnement similaire à la bascule RS à portes NON-OU mène au tableau de Karnaugh et à l'expression de la sortie Q suivants :

Q_n \ RS	00	01	11	10
0	0	1	1	0
1	1	1	1	0

$$Q_{n+1} = R + \overline{S}.Q_n$$

Aspect asynchrone

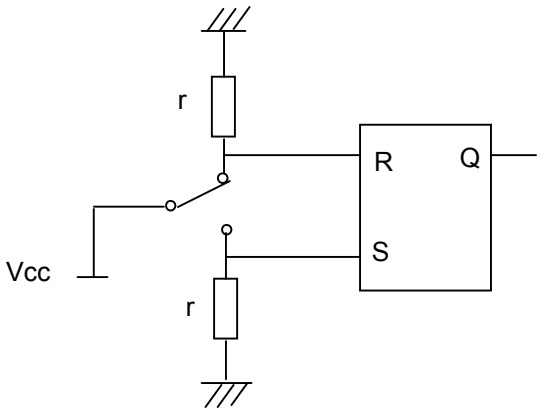
Les 2 bascules RS étudiées ci-dessus sont asynchrones, c'est à dire qu'il n'existe pas de signal par rapport auquel la mise à jour des sorties serait synchronisée ; dès qu'un changement apparaît en entrée, les sorties sont mises à jour immédiatement (plus exactement, après une durée très faible correspondant au temps de propagation des portes).

Exemple d'application : circuit anti-rebonds

Une des applications classiques de la bascule RS est le circuit anti-rebonds. Il s'agit de pallier à un inconvénient d'un commutateur mécanique dans une commande numérique. Le passage d'un niveau logique à l'autre (0 à 1 ou 1 à 0) ne se produit jamais "proprement" ; il y a toujours des rebonds mécaniques, qui font qu'il existe une alternance de niveaux 0 et 1 avant stabilisation (tout ceci se produisant sur quelques ms).

Une solution est donc d'utiliser une bascule RS, qui va se déclencher dès le premier changement d'état, et rester bloquée tant que la commande opposée n'aura pas été effectuée.

On peut avoir par exemple le schéma suivant :



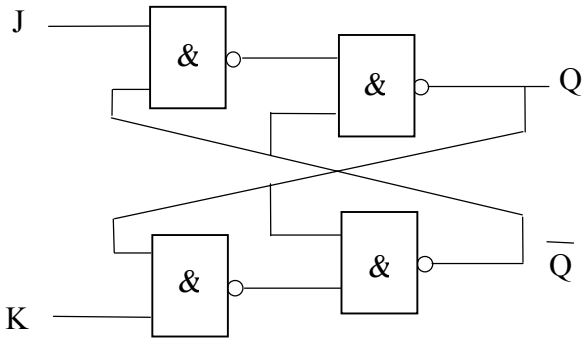
Quand le commutateur est en position haute, R=1 et S=0 donc Q=0. Quand il change de position, on passe par l'état R=S=0, donc on est dans l'état de mémorisation et les sorties ne changent pas. Quand il est en position basse, on a R=0 et S=1, donc Q=1.

On voit que la bascule ne passe jamais par un état indéterminé, et que s'il y a des rebonds, ils ne sont pas gênants puisqu'on reste dans l'état de mémorisation jusqu'à ce que l'autre entrée passe à 1.

b) Bascule JK

La bascule JK est dérivée d'une bascule RS (à portes NON-ET), avec J et K tels que:

$$R = J.\overline{Q} \qquad \text{et} \qquad S = K.Q$$



Par rapport à la bascule RS, l'intérêt est que l'état inutilisé devient utilisable. En effet, à la différence de la bascule RS, si J=K=1, les 2 sorties ne sont pas égales toutes les 2 à 1. Quand on arrive dans l'état J=K=1 à partir de l'un des 3 autres, on a donc forcément les 2 sorties complémentaires.

Par contre, quand au moins l'une des 2 entrées J et K est égale à 0, on retrouve le même comportement que la RS classique, car, par exemple pour K :

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

(le fait que K soit combiné 2 fois de suite avec Q au moyen d'une porte NON-ET équivaut à une seule combinaison).

Le schéma est alors équivalent à celui de la bascule RS à portes NON-ET.

La table de vérité de la bascule JK est donc :

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

Le tableau de Karnaugh correspondant est :

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

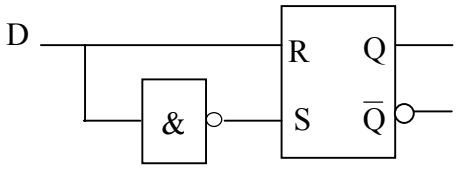
Son équation logique est donc :

$$Q_{n+1} = J.\overline{Q_n} + \overline{K}.Q_n$$

La combinaison d'entrée J=K=1 devient utilisable, mais elle pose un nouveau problème : puisque dans ce cas Q recopie $\overline{Q}$, cette recopie n'a aucune raison de ne se produire qu'une seule fois. En pratique, la sortie Q oscille entre 0 et 1, à la fréquence imposée par le temps de propagation des portes logiques, pendant tout le temps où cette combinaison est présente en entrée.

c) Bascule D

Supposons que l'on contraigne les entrée R et S de la bascule pour qu'elles soient toujours complémentaires, avec R=D et $S = \overline{D}$:



Seules 2 lignes de la table de vérité de la bascule RS sont alors utilisées :

R	S	Q_{n+1}
~~0~~	~~0~~	~~Q_n~~
0	1	0
1	0	1
~~1~~	~~1~~	~~non utilisé~~

On peut la simplifier de la manière suivante :

D	Q_{n+1}
0	0
1	1

On en déduit la relation entrée-sortie de cette bascule :

$$Q_{n+1} = D$$

Sous cette forme, la bascule D n'a pas grand intérêt puisqu'il s'agit d'un bloc fonctionnel qui ne fait que recopier son entrée, en permanence. On verra dans le paragraphe suivant que son utilité apparaît avec un signal supplémentaire dit "d'horloge" (version synchrone).

On peut également réaliser une bascule Dà partir d'une bascule JK, ce qui revient au même au niveau de son fonctionnement. Il suffit de poser $J = \overline{D}$ et K=D.

d) Bascule T

Si l'on relie ensemble les 2 entrées d'une bascule JK, on obtient ce que l'on appelle une bascule T. On limite alors son fonctionnement à 2 lignes de la table de vérité de la bascule JK (la 1ère et la 4e).

On a alors le fonctionnement suivant :

- si T=0, $Q_{n+1}=Q_n$: état de mémorisation ;
- si T=1, $Q_{n+1} = \overline{Q_n}$: changement d'état.

Ainsi, si T est un signal carré, Q sera également un signal carré, de fréquence 2 fois moindre, synchronisé sur T.

Son équation logique s'écrit :

$$Q_{n+1} = T.\overline{Q_n} + \overline{T}.Q_n = T \oplus Q_n$$

III.1.2) Bascules synchrones

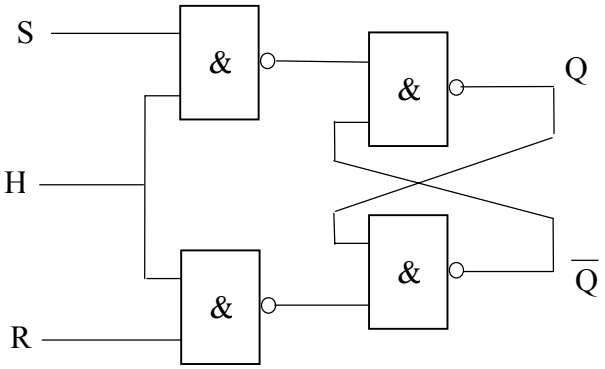
La synchronisation des circuits séquentiels peut se faire de trois façons différentes :

- sur niveau : les changements d'état ont lieu pour un niveau donné (0 ou 1) de l'horloge ;
- sur front : les bascules changent d'état uniquement sur un front montant ou descendant de l'horloge ;
- par impulsion : les bascules changent d'état après deux fronts successifs de l'horloge (front montant puis descendant ou vice versa).

a) Synchronisation sur niveau

Bascule RS synchrone (RSH)

Reprenons notre bascule RS à portes NAND, et conditionnons les 2 entrées à une 3e, appelée H :



Si H=0, cela est équivalent à avoir R=S=0. On est donc dans l'état de mémorisation, et ceci quels que soient les états de R et de S.

Par contre, si H=1, le fonctionnement revient à celui de la bascule RS à portes NON-ET, étudié précédemment.

L'entrée H permet donc de bloquer ou non le fonctionnement de la bascule RS. Il est utilisé comme signal de synchronisation. En général, il est périodique ; on l'appelle signal d'horloge. Il permet de n'autoriser les changements de la sortie qu'à des instants bien précis. On peut donc synchroniser le fonctionnement de cette bascule sur d'autres circuits.

Ici, cette synchronisation s'effectue sur un niveau.

Cette bascule est appelée également RSH ou RST.

Pour obtenir la table de vérité de cette bascule, il suffit de reprendre celle de la bascule RS à portes NON-ET :

$$Q_{n+1} = R + \overline{S}.Q_n$$

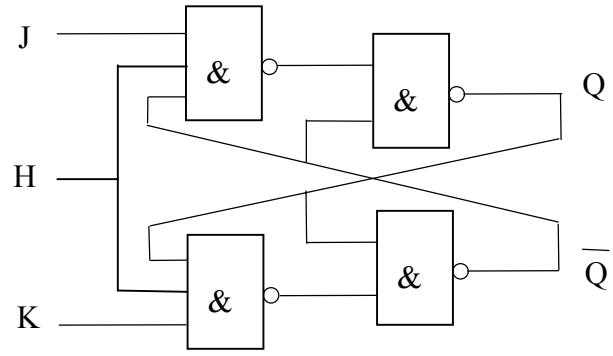
et de remplacer R par R.H et S pas S.H :

$$Q_{n+1} = R.H + \overline{S.H}.Q_n$$

Bascule JK synchrone (ou JKH)

Il s'agit de la bascule JK étudiée précédemment, à laquelle une entrée de synchronisation est ajoutée.

Le schéma du circuit correspondant est le suivant :



On utilise les symboles suivants :

Le symbole de gauche correspond à une synchronisation sur niveau haut (=1), celui de droite sur niveaubas (= 0).

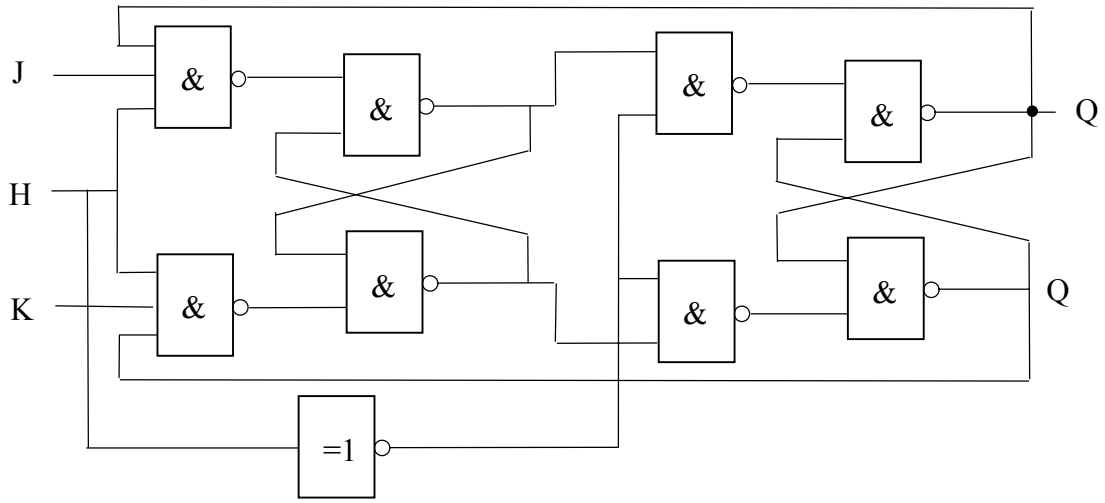
Son équation logique est :

$$Q_{n+1} = J.H.\overline{Q_n} + \overline{K.H}.Q_n$$

On retrouve la limitation de la bascule JK asynchrone pour H=J=K=1 : la sortie Q oscille entre 0 et 1 pendant toute la durée de l'état haut du signal d'horloge.

Pour y remédier, on peut utiliser une bascule JK Maître-Esclave. Elle consiste à utiliser 2 bascule JK en cascade, la 2^{e} recevant comme signal d'horloge le signal d'horloge de la 1ère, complémenté. De plus, le bouclage des sorties de la 2^{e} bascule (en partant de la gauche) s'effectue sur les portes d'entrée de la 1ère. Ainsi, le retour d'information ne s'effectue qu'après un niveau haut suivi par un niveau bas sur H. Cela revient à obtenir un déclenchement sur impulsion.

Le schéma de cette bascule est le suivant :

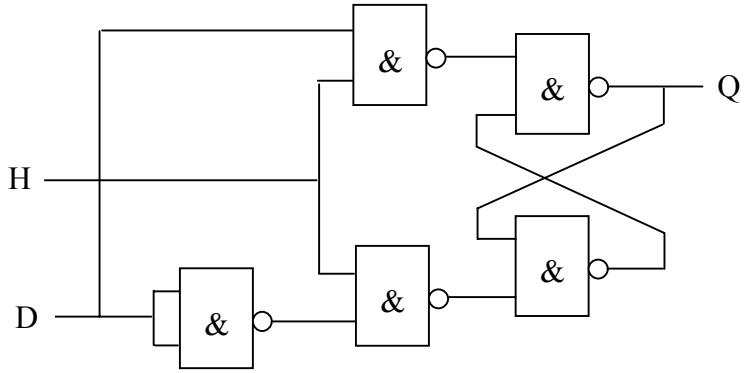


La bascule maître-esclave est donc une solution à l'oscillation existant pour la combinaison J=K=1. Une autre solution consiste à utiliser un déclenchement sur front (voir bascules synchrones, plus loin).

Bascule D à verrou (D latch)

On a vu qu'une bascule D recopiait sur sa sortie Q son entrée D. Avec une entrée supplémentaire d'horloge, on peut synchroniser la recopie sur cette dernière. On prend une bascule RSH, dans laquelle on complémente les entrées.

La bascule recopie l'entrée D en sortie Q quand l'horloge est active, c'est à dire sur niveau haut. Quand l'horloge est inactive, la bascule garde l'état précédent : $Q_{n+1}= Q_n$.



On utilise les schémas de blocs fonctionnels suivants, pour une synchronisation sur niveau haut (gauche) et niveau bas (droite) :

Pour la synchronisation sur niveau haut, l'équation logique associée est :

$$Q_{n+1} = D.H + \overline{H}.Q_n$$

Cette bascule est très utilisée dans les compteurs synchrones.

Bascule T synchrone

Une bascule T synchrone est une bascule T asynchrone possédant une entrée supplémentaire, selon le même principe que les bascules JK et D synchrones.

Son équation logique est donc :

$$Q_{n+1} = (T.H) \oplus Q_n$$

b) Synchronisation sur front

Les bascules synchrones sur front changent d'état uniquement sur un front du signal d'horloge ; en dehors de ces fronts, elle fonctionne en mémoire.

Ce front peut être montant ou descendant. On utilise respectivement les 2 symboles suivants :

Ce mode de fonctionnement protège d'éventuels parasites sur les entrées car les entrées ne sont prises en compte que pendant la durée d'un front, qui est très courte.

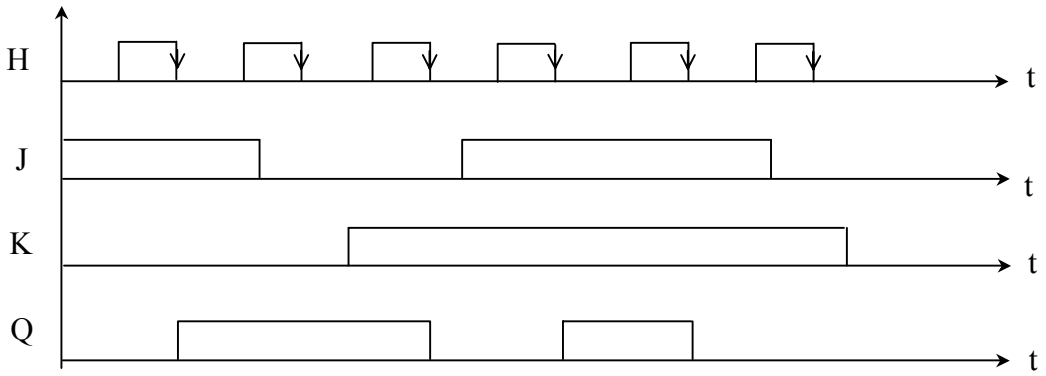
Bascule J K à déclenchement sur front (edge triggered)

La table de fonctionnement d'une bascule JK à déclenchement sur front est la suivante :

H	J	K	Q_{n+1}
↑ ou ↓	0	0	Q_n
↑ ou ↓	0	1	0
↑ ou ↓	1	0	1
↑ ou ↓	1	1	$\overline{Q}_n$

On utilise les mêmes symboles que pour les bascules à déclenchement sur niveau. Le cercle devant H indique un déclenchement sur front descendant, l'absence de cercle un déclenchement sur front montant :

Le chronogramme suivant est un exemple de fonctionnement d'une bascule J K sur front descendant, avec au départ Q=0 :



Bascule D à déclenchement sur front (edge triggered)

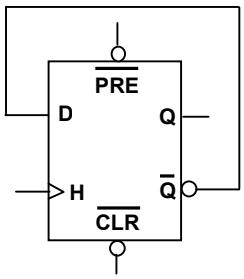
Dans une bascule D à déclenchement sur front, l'entrée D n'est prise en compte qu'au front montant (ou descendant) de l'horloge.

Les symboles de cette bascule à déclenchement sur front montant et sur front descendant sont respectivement :

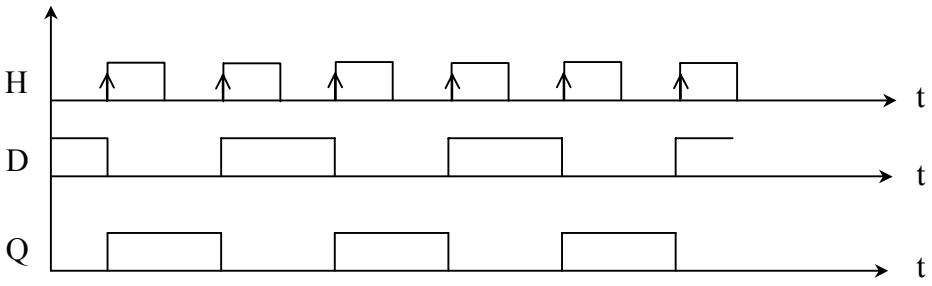
Quand il y a un front actif la sortie recopie l'entrée sinon la sortie garde son état précédent.

Application à la division de fréquence

Considérons le circuit suivant, dans lequel la sortie complémentée est rebouclée sur l'entrée D.



Supposons que Q est au départ à 0. Traçons le chronogramme de ces signaux pour 6 impulsions d'horloge :



Au départ, Q=0 donc $D = \overline{Q} = 1$. Au 1er front montant d'horloge, la sortie Q recopie la valeur de D (=1) et $\overline{Q}$ passe à 0. Au 2e front montant d'horloge, la sortie Q recopie de nouveau la valeur de D (=0) et $\overline{Q}$ passe à 1, et ainsi de suite.

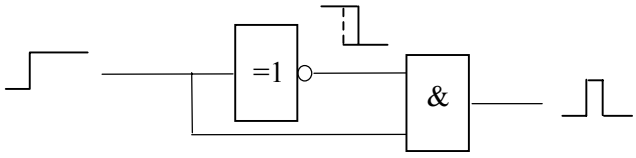
On constate que la sortie Q possède une fréquence 2 fois plus petite que celle de l'horloge. On a donc réalisé un diviseur de fréquence par 2.

Cette propriété est à la base des compteurs.

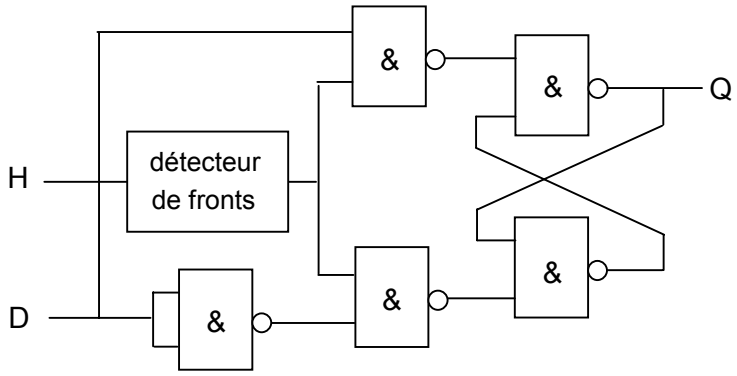
De même, si l'on place les entrée J et K d'une bascule JK à 1, la sortie changera d'état à chaque front actif d'horloge. Le signal en sortie aura une fréquence divisée par deux par rapport à celle de l'horloge.

Détection des fronts

Pour réaliser une synchronisation sur front, il faut utiliser un circuit détecteur de fronts. Le circuit suivant permet de générer une impulsion de courte durée, par exploitation du temps de propagation des portes logiques :



En insérant ce détecteur de fronts en entrée d'une bascule, on transforme une bascule asynchrone en bascule synchrone. Par exemple, pour une bascule D :



c) Entrées de forçage

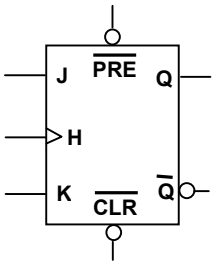
Les entrées de bascules que nous avons étudiées jusqu'ici sont appelées entrées synchrones car les changements des sorties qu'elles provoquent sont synchronisés sur un signal d'horloge H.

La plupart des bascules sont également munie d'entrées asynchrones appelées entrées de forçage ou de pré-positionnement, prioritaires et ne dépendant pas de l'horloge.

Il existe deux types de forçage :

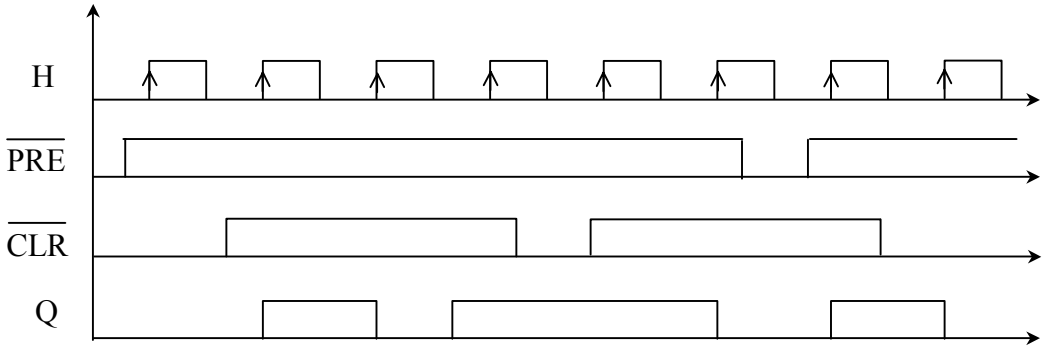
- mise à 1 : SET ou PRESET ou RAU (Remise A Un) ;
- mise à 0 : RESET ou CLEAR ou RAZ (Remise A Zéro).

Sur le schéma des bascules, elles sont en général situées en haut et en bas du bloc fonctionnel, comme le sur la figure suivante :



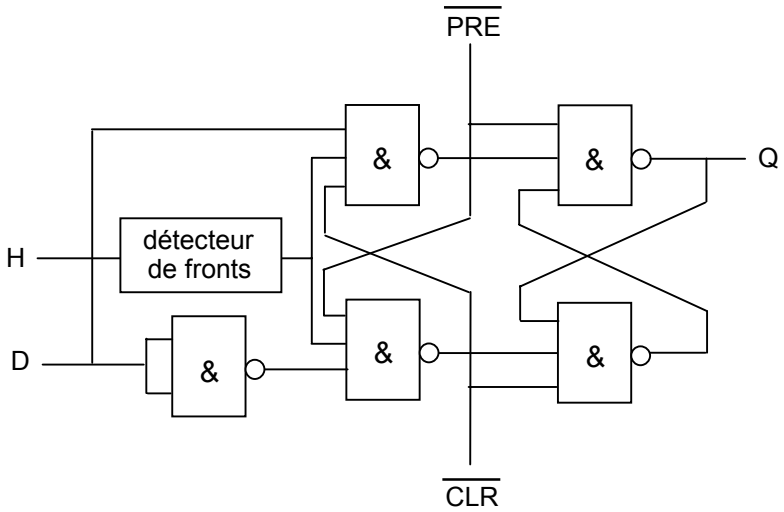
Elles sont généralement actives à l'état bas. Par exemple, un 0 sur l'entrée CLR provoque une remise à zéro de la bascule quelles que soient les valeurs des entrées synchrones. Ces deux entrées asynchrones ne peuvent être actives (=0) en même temps.

Voici un exemple de chronogramme de fonctionnement, pour J=K=1 :



Quand les entrées asynchrones sont inactives (=1), la bascule JK fonctionne en diviseur de fréquence par deux (J=K=1 : la sortie change d'état à chaque front actif de l'horloge).

Ces entrées de forçage sont simplement des entrées supplémentaires des portes situées en sortie. Par exemple, dans le cas d'une bascule D :



La barre sur le PRESET et sur le CLEAR indiquent que ces 2 entrées sont actives au niveau bas.

Quand ces 2 entrées sont au niveau 1, elles sont inactives puisque le 1 est l'élément neutre du ET.

Quand le PRESET est à 0 et le CLEAR à 1, un niveau 1 est imposé sur la sortie Q, et un niveau 0 sur la sortie $\overline{Q}$.

Quand le CLEAR est à 0 et le PRESET à 1, un niveau 0 est imposé sur la sortie Q, et un niveau 1 sur la sortie $\overline{Q}$.

Quand le CLEAR et PRESET sont tous les deux à 0, le fonctionnement de la bascule n'est plus correct puisqu'on a $Q = \overline{Q} = 1$..

d) Tables de transition

La table de transition donne les états dans lesquels doivent se trouver les entrées pour obtenir chacune des 4 transitions possibles de la sortie Q. Quand l'état de l'entrée considérée est indifférent (0 ou 1), on le remplace par une croix.

Bascule JK

Rappelons la table de vérité d'une bascule JK :

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Par exemple, pour obtenir la transition 0→1 d'une sortie Q, il faut avoir

- J=K=1 qui inverse l'état de la bascule, ou
- J=1 et K=0 qui force la sortie de la bascule à 1.

Dans les 2 cas, il faut J=1, quel que soit l'état de l'entrée K. Pour cette transition, on doit donc mettre l'entrée J à 1, l'état de K étant indifférent.

De la même manière, on détermine la table de transition complète :

Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Par exemple, pour réaliser un diviseur de fréquence par 2, on veut que la sortie de la bascule change d'état à chaque front actif d'horloge. On cherche les valeurs de J et K pour lesquelles on à les transitions 0→1 et 1→0.

Dans les 2 lignes correspondantes de la table de transition, les 2 entrées ne sont jamais spécifiées simultanément ; il est donc possible de choisir, pour simplifier, l'égalité des 2 entrées :

$$J=K$$

Bascule D

Rappelons la table de vérité d'une bascule D :

D	Q_{n+1}
0	0
1	1

La détermination de sa table de transition utilise la même méthode que pour la bascule JK, mais elle est plus simple dans la mesure où il n'y a pas vraiment d'état de mémorisation. Il suffit donc de recopier les valeurs de la sortie Q_{n+1} pour obtenir les valeurs correspondantes de l'entrée :

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Par exemple, pour obtenir un diviseur par 2, on veut que la sortie Q change d'état à chaque front actif d'horloge. Ceci correspond aux 2^e et 3^e lignes de la table de transition. Pour ces 2 lignes, on constate que l'on a $D = \overline{Q}_n$. Il suffit donc de relier la sortie $\overline{Q}_n$ à l'entrée D.

e) Exigences de synchronisation

Pour que le fonctionnement d'une bascule synchrone puisse être fiable, il y a 2 conditions à respecter au niveau des durées des signaux. Il existe

- la **durée de stabilisation** : avant présentation du niveau ou du front actif de l'horloge, c'est la durée minimale depuis laquelle l'entrée est présente ;
- la **durée de maintien** : c'est la durée minimale pendant laquelle l'entrée doit être présente après la présentation du niveau ou du front actif de l'horloge.

III.2) Registres

Un registre est un ensemble de bascules, synchronisées par la même horloge.

Les registres sont à la base d'opérations couramment utilisées dans les ordinateurs : mémorisation provisoire (mémoires-tampon), décalages, rotations, etc.

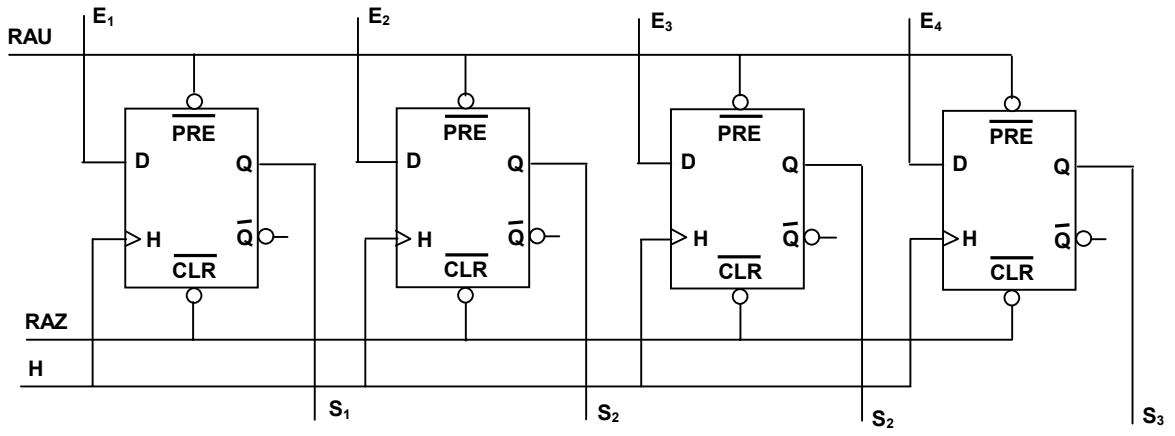
III.2.1) Différents types de registres

On distingue quatre types de registres selon la façon dont sont utilisées les entrées et les sorties : en parallèle ou en série.

a) Registres à entrées parallèles, sorties parallèles

Toutes les entrées (E_1, E_2, E_3, E_4) sont introduites en même temps dans le registre. Toutes les sorties (S_1, S_2, S_3, S_4) sont disponibles au même instant. Les signaux RAZ et RAU sont des entrées asynchrones permettant respectivement la remise à zéro ou la remise à un de toutes les bascules en même temps.

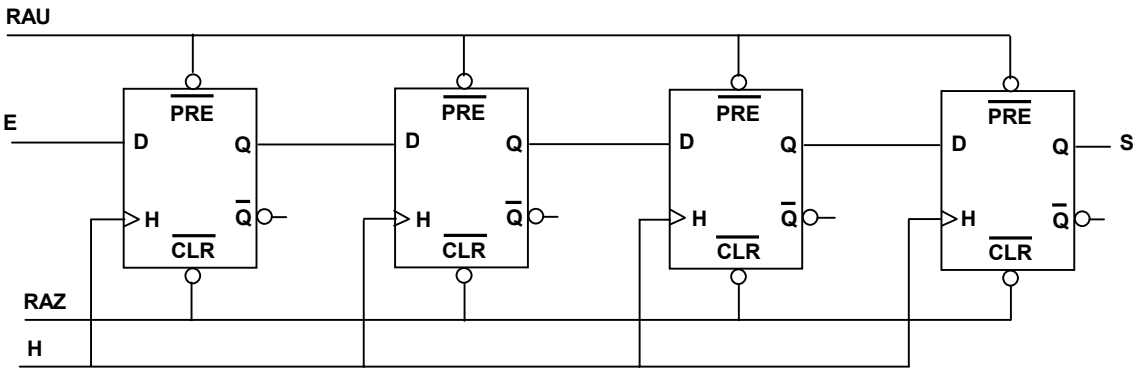
On considère un registre de quatre bits. Les bascules utilisées dans les exemples suivants sont des bascules D mais un registre peut également être réalisé à partir de bascules JK.



Ce type de registre est aussi appelé registre tampon. Il est souvent utilisé pour la mémorisation de données de durée brève ou pour le transfert de données.

b) Registres à entrée série, sortie série

Ce registre possède une entrée E et une sortie S. Les données binaires d'entrée sont introduites bit après bit. Elles sont également disponibles les unes après les autres au rythme de l'horloge en sortie. Ce type de registre est utilisé pour effectuer des décalages.

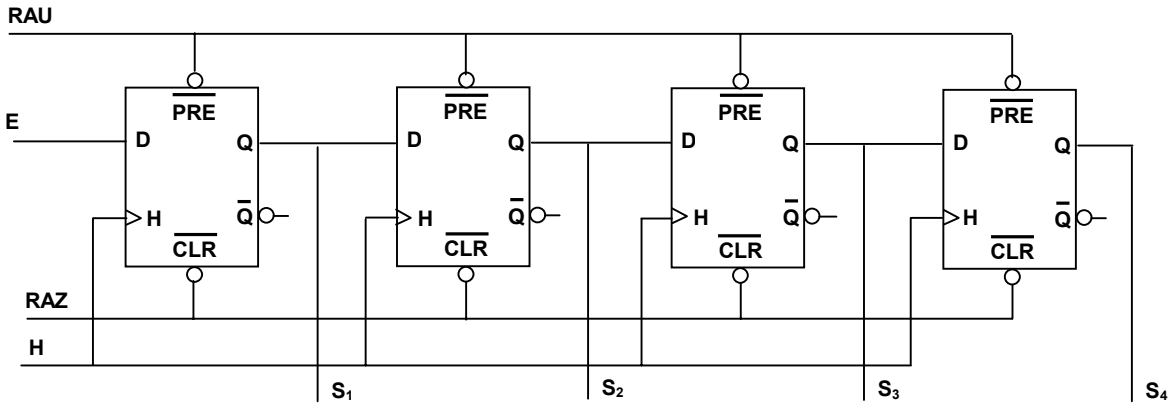


En rebouclant la sortie de la dernière bascule sur l'entrée de la première, on obtient ce qu'on appelle un "**compteur en anneau**". Pour charger une donnée 4 bits initiale sur les entrées D des bascules, il faut ajouter une logique de commande composée de quelques portes supplémentaires. Cette donnée se retrouver cycliquement sur les mêmes bascules.

En rebouclant la sortie complémentée $\overline{Q}$ de la dernière bascule sur l'entrée de la première, on obtient ce que l'on appelle un "**compteur Johnson**". Ce compteur possède un modulo égal à 2n, où n est le nombre de bascules.

c) Registres à entrée série, sorties parallèles

Ce registre possède une entrée E et plusieurs sorties (S_1, S_2, S_3, S_4). Les données binaires d'entrée sont introduites bit après bit. Les sorties sont toutes disponibles en même temps. Ces registres peuvent être utilisés pour faire une transformation série-parallèle des données.

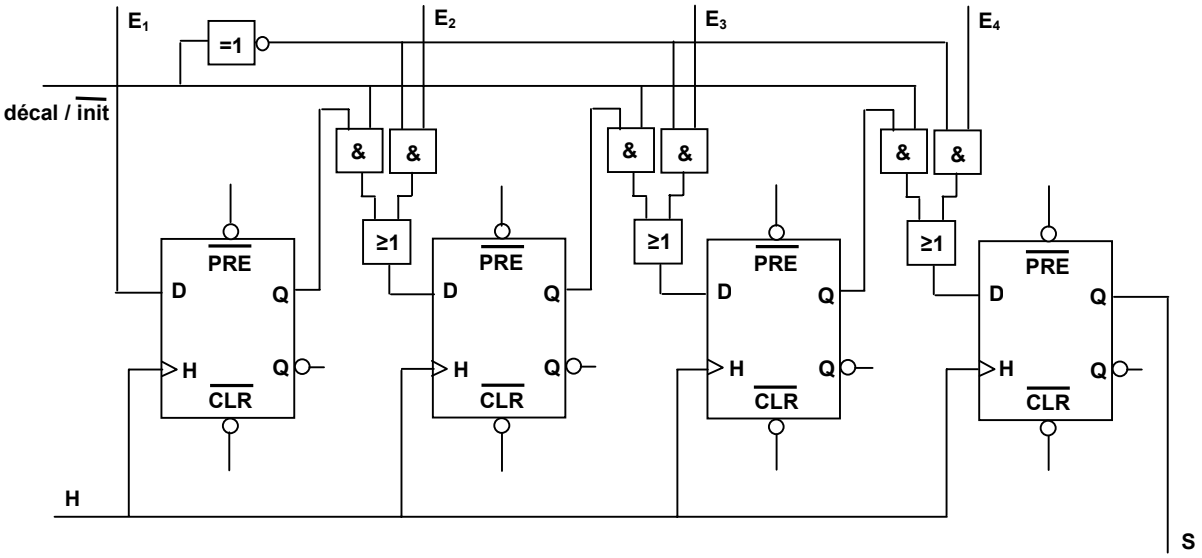


La donnée est disponible après N coups d'horloge, où N est le nombre de bascules.

d) Registres à entrées parallèles, sortie série

Toutes les entrées (E_1, E_2, E_3, E_4) sont introduites en même temps dans le registre. Les informations en sortie sur S sont disponibles les unes après les autres au rythme de l'horloge. Ces registres peuvent être utilisés pour faire une transformation parallèle-série des données.

La sortie Q d'une bascule est reliée à l'entrée D de la bascule suivante. Les entrées parallèles ne peuvent pas être appliquées directement sur les entrées des bascules, puisqu'elles mettraient en court-circuit les sorties des bascules précédentes. Il faut utiliser une logique de commande à base de portes logiques ET et OU, ayant pour signal d'entrée une commande de chargement/décalage.

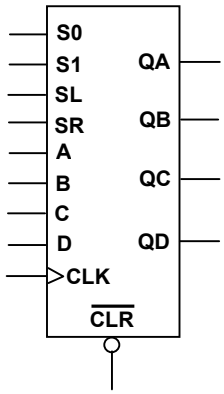


III.2.2) Registres universels

Il existe des circuits intégrés regroupant les quatre types de registres présentés ci-dessus. Ils permettent les modes de fonctionnement suivants :

- chargement et lecture parallèles,
- chargement série et décalages à droite ou à gauche, lecture série ou parallèle,
- chargement parallèle et décalages à droite ou à gauche, lecture série ou parallèle.

Par exemple, le circuit intégré de référence 74194 possède la représentation symbolique suivante :



Les entrées A, B, C, D sont les entrées parallèles. Les entrées SL et SR sont respectivement les entrées/sorties série gauche et droite. Les entrées S0 et S1 permettent de choisir le mode de fonctionnement de ce registre (blocage, décalage à droite, décalage à gauche, chargement parallèle). L'entrée CLR (active sur niveau bas) permet une remise à zéro asynchrone des sorties. L'entrée CLK est l'entrée horloge de synchronisation. Les sorties sont QA, QB, QC, QD.

III.2.3) Application des registres

Un registre permet de mémoriser de façon temporaire un mot de N bits en attendant son traitement ultérieur. Chacune de ses bascules permet de mémoriser 1 bit.

Les registres sont utilisés dans les microprocesseurs pour des mémorisations temporaires des données.

Un registre de huit bits est constitué de 8 bascules synchronisées par la même horloge. Un registre est un circuit permettant :

- de stocker une information binaire,
- de transférer une information dans certaines conditions,
- de faire des traitements simples sur les éléments binaires comme des décalages ou des rotations.

Les systèmes numériques utilisent des mots ou des nombres de n éléments binaires. On a souvent besoin d'avoir accès à un élément binaire ou un groupe d'éléments binaires. On utilise, pour cela, des fonctions de décalage des éléments binaires.

Soit un mot de huit éléments binaires (constitué de huit bascules), schématisé par :

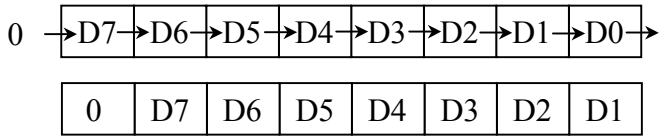
D7	D6	D5	D4	D3	D2	D1	D0

On peut effectuer différents types de décalages sur ce mot.

a) Décalage

Décalage à droite

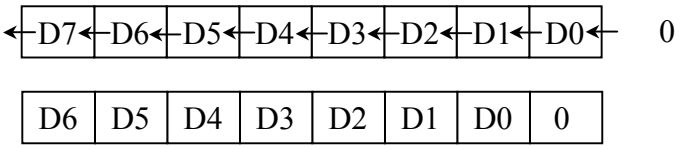
Tous les éléments binaires sont décalés d'un rang vers la droite ; il apparaît un 0 (ou un 1) sur l'élément binaire de poids fort (bit le plus à gauche). L'élément binaire de poids faible (le bit le plus à droite) est perdu.



Si la valeur 0 est entrée sur l'élément binaire de poids fort, on obtient une division par deux du nombre initial.

Décalage à gauche

Tous les éléments binaires sont décalés d'un rang vers la gauche ; il apparaît un 0 (ou un 1) sur l'élément binaire de poids faible (bit le plus à droite). L'élément binaire de poids fort (le bit le plus à droite) est perdu.



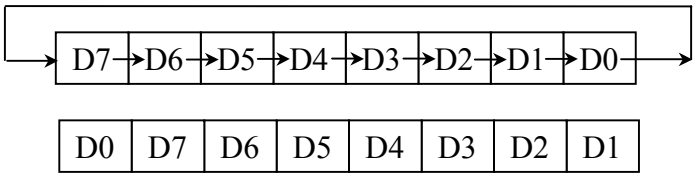
Si la valeur 0 est entré sur l'élément binaire de poids faible, on obtient une multiplication par deux du nombre initial.

b) Rotation

Une rotation est un décalage circulaire.

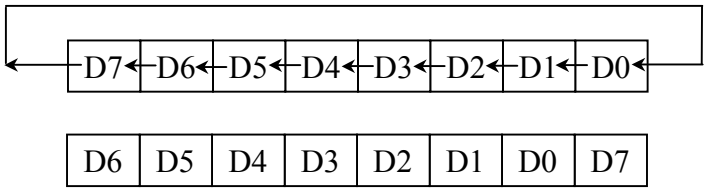
Rotation à droite

Tous les éléments binaires sont décalés vers la droite et le bit de poids fort prend la valeur du bit de poids faible.



Rotation à gauche

Tous les éléments binaires sont décalés vers la gauche et le bit de poids faible prend la valeur du bit de poids fort.



III.3) Compteurs

Il est possible de connecter des bascules pour effectuer des opérations de comptage. Le nombre de bascules utilisées et la façon dont elle sont interconnectés déterminent le nombre d'états du compteur. L'état du compteur est défini par le nombre binaire formé par l'ensemble des sorties des bascules.

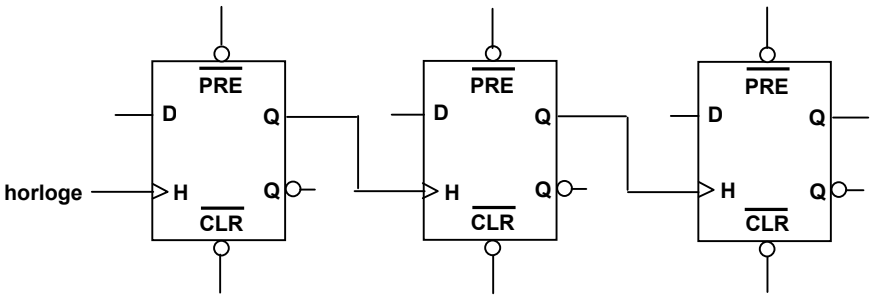
Les compteurs sont des éléments essentiels de la logique séquentielle car ils sont utilisés pour les systèmes arithmétiques, mais ils permettent aussi d'établir une relation d'ordre de succession d'événements. Ils sont utilisés notamment pour le comptage d'événements, pour diviser la fréquence, dans les automates programmables, etc.

Les compteurs sont classés en deux catégories selon leur mode de fonctionnement. On distingue les compteurs asynchrones des compteurs synchrones.

Les compteurs asynchrones ou compteurs série

La première bascule est synchronisée par une horloge externe mais le déclenchement des autres bascules est déterminé par une combinaison logique des sorties des bascules précédentes. La propagation de l'ordre de changement d'état se fait en cascade. Les sorties des bascules ne changent pas d'état exactement en même temps car elles ne sont pas reliées au même signal d'horloge.

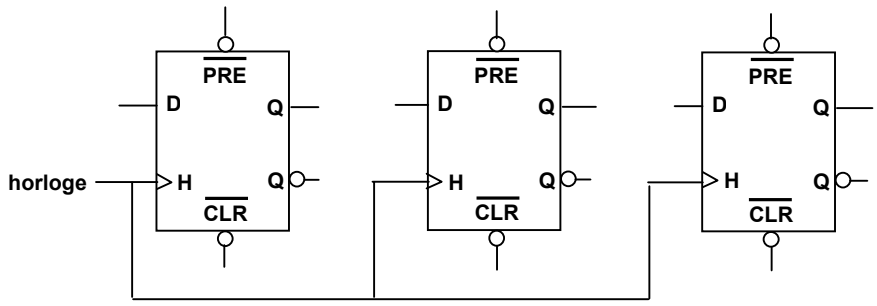
Le schéma ci-dessous est un exemple de connexion d'horloge pour obtenir un compteur asynchrone avec propagation de l'ordre de changement d'état en cascade (les entrées de donnée D étant en l'air, le schéma n'est pas complet).



Les compteurs synchrones ou parallèles

Le signal d'horloge externe est connecté à toutes les bascules et permet de les déclencher toutes simultanément.

Le schéma ci-dessous est un exemple de connexion de l'horloge pour obtenir un compteur synchrone : la synchronisation des bascules est faite par le même signal d'horloge.



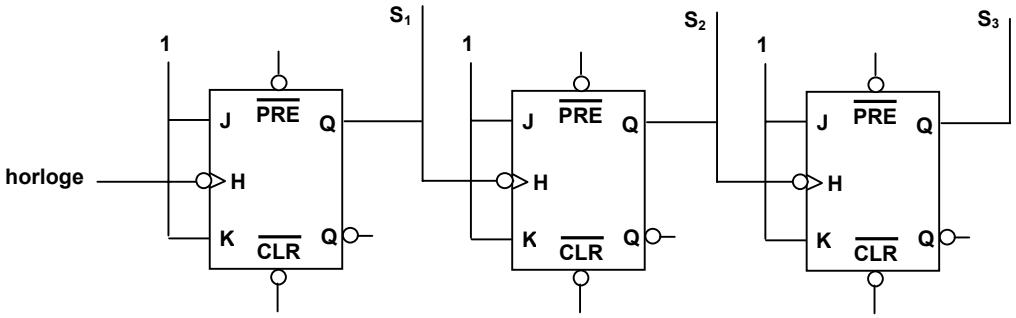
Pour chacune de ces catégories, les compteurs sont classés selon leurs séquences, le nombre d'états ou le nombre de bascules qu'ils comportent.

III.3.1) Compteurs asynchrones

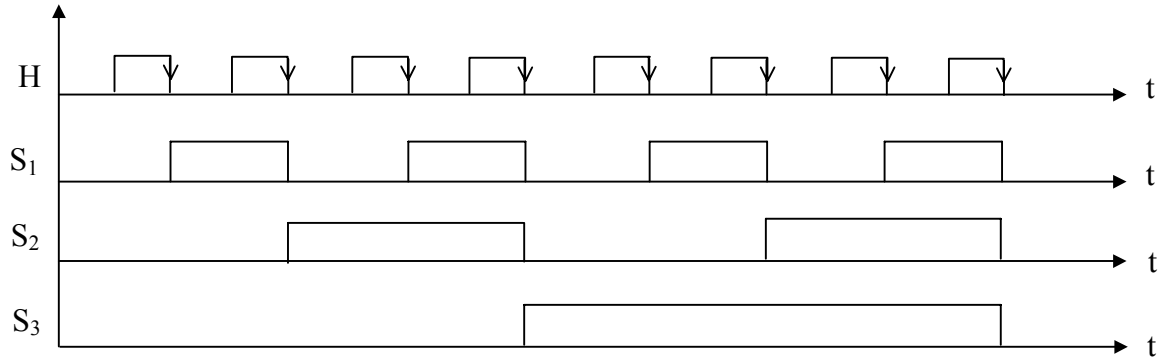
a) Compteur binaire

On a vu qu'en connectant les entrées J et K d'une bascule JK à 1, la sortie Q de cette bascule constituait la fréquence d'horloge divisée par 2.

Le schéma ci-dessous représente un exemple de compteur à 3 bascules JK.



Les entrées de pré-positionnement sont inactives car elles sont placées au niveau haut. Les entrées J et K sont à 1 donc les bascules changent d'état à chaque front actif de leur horloge (front descendant). La bascule S_1 change d'état à chaque front descendant de l'horloge externe, la bascule S_2 à chaque front descendant de S_1 et la bascule S_3 à chaque front descendant de S_1.



On affecte les poids 1, 2, 4 aux sorties S_1, S_2, S_3 des bascules. On considère qu'au départ toutes les sorties sont à zéro. On obtient le tableau suivant (il y a un changement d'état après chaque front descendant de l'horloge).

front actif de l'horloge	S_3	S_2	S_1	État
état initial	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

b) Compteur modulo N

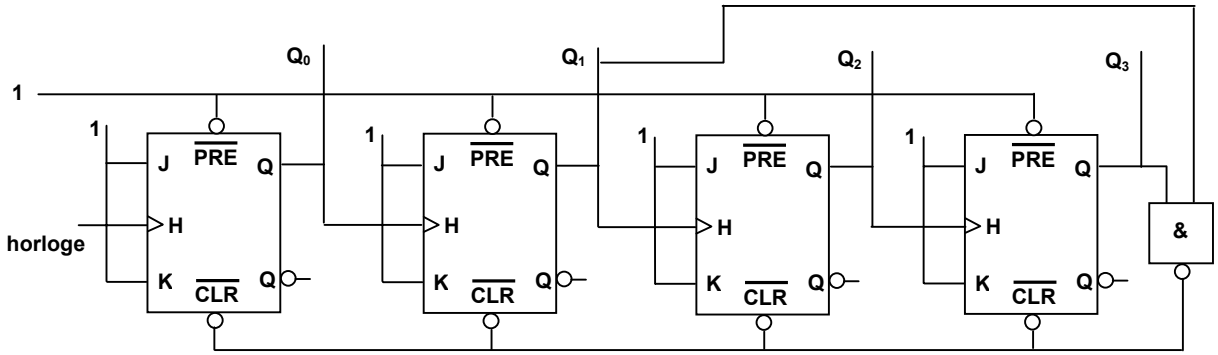
On appelle compteur modulo N, un compteur qui compte de 0 à N-1. Le compteur le plus utilisé est le compteur modulo 10 ou compteur à décade ou compteur DCB (ou BCD). Il produit une séquence de 0 à 9.

Pour réaliser un tel compteur, on va prendre un compteur binaire et tronquer sa séquence de sortie en effectuant une remise à 0 de toutes ses sortie de façon prématurée. On choisit donc la puissance de deux immédiatement supérieure à la longueur de la séquence : $16=2^4$; il faut 4 bascules pour réaliser le compteur et on va utiliser l'entrée de remise à zéro asynchrone pour tronquer la séquence à 9. Le compteur comptera donc de 0 (0000) à 9 (1001). Le principe est identique pour réaliser n'importe quel compteur modulo N.

Quand le compteur passe à l'état 1010 (10), c'est à dire quand B=D=1, on provoque une remise à zéro asynchrone du compteur par l'intermédiaire d'une porte NON-ET car l'entrée CLR est active au niveau bas.

Ce compteur reste dans l'état transitoire non souhaité 1010 (10) pendant le temps de traversé de la porte NON-ET (quelques nano-secondes), puis est ensuite remis à zéro. Pour fonctionner correctement les temps de propagation de toutes les portes doivent être identiques pour obtenir en même temps l'état 1010.

La figure ci-dessous représente un compteur décimal à base de bascule J K :



c) Inconvénients et avantages des compteurs asynchrones

Les compteurs asynchrones sont assez lents car les temps de propagation de chaque bascule s'ajoutent.

La propagation des signaux de déclenchement des bascules provoque des états transitoires qui sont indésirables quand ils sont présents durant un temps non négligeable.

Par contre, la conception de ces compteurs est très simple et les liaisons entre les bascules sont peu nombreuses.

La méthode décrite ci-dessus permet de réaliser des compteurs ou décompteurs, mais pas des séquences quelconques.

III.3.2) Compteurs synchrones

Dans les compteurs synchrones, toutes les bascules reçoivent le même signal d'horloge. Ils permettent d'éliminer les problèmes temporels dus à l'accumulation des temps de propagation des bascules.

a) Détermination directe des entrées des bascules

La conception d'un compteur consiste d'abord à établir la table de vérité recherchée.

Ensuite, il faut déterminer les combinaisons des entrées des bascules permettant d'obtenir cette table de vérité.

Prenons l'exemple d'un compteur binaire par 8, réalisé à l'aide de bascules JK. On établit d'abord sa table de vérité :

front actif de l'horloge	Q_2	Q_1	Q_0	État
état initial	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

On choisit d'utiliser des bascules avec entrée d'horloge actives sur front montant. On choisit également de relier J et K pour chacune des bascules. On sait que l'on a alors 2 cas possibles :

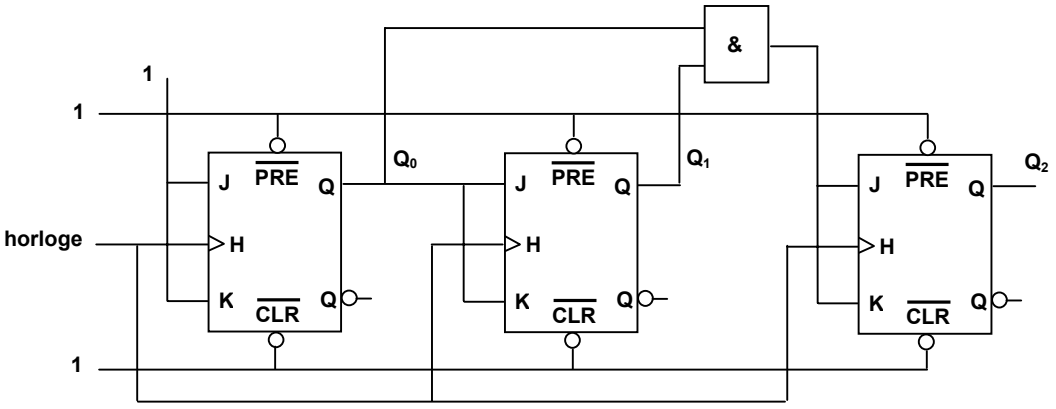
- 1er cas : J=K=0 ; au coup d'horloge, la sortie de la bascule ne change pas d'état ;
- 2^{e} cas : J=K=1 ; au coup d'horloge, la sortie de la bascule change d'état.

On constate que la bascule Q_0 change d'état à chaque front d'horloge, on l'utilise donc comme diviseur par 2 : on a vu précédemment qu'il fallait avoir pour cela $J_0=K_0=1$.

Pour les autres bascules, on doit regarder l'état des autres sorties quand elles changent d'état. On remarque que la bascule Q_1 change d'état à chaque fois que Q_0 passe de 1 à 0, et ceci indépendamment de Q_2. Il suffit donc de relier Q_0 à J_1 et K_1. On posera donc $J_1=K_1=Q_0$.

De même, la bascule Q_2 change d'état après chaque fois que $Q_1=Q_0=1$. Il suffit d'utiliser cette condition pour avoir J_1 et K_1 à 1 au moment du coup d'horloge adéquat. On posera donc $J_1=K_1=Q_0.Q_1$.

Finalement, le schéma du circuit est le suivant :



b) Utilisation des tables de transition

Pour déterminer les combinaisons d'entrée des bascules de manière systématique, on peut utiliser la table de transition de la bascule utilisée.

Contrairement à la table de vérité qui donne les valeurs des sorties en fonction des entrées, la table de transition indique la valeur des entrées pour une transition donnée en sortie. Il y a 4 transitions possibles en sortie.

Avec des bascules JK

Pour une bascule JK, voici le rappel de sa **table de transition** :

Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

On définit la **table des états futurs**, qui indique l'état futur pour tous les états possibles des sorties du compteur.

Ici il y a 3 bascules ; il y a donc 8 lignes dans cette table. On en déduit l'état des entrées J et K, à partir de la table de transition :

$Q_2(n)$	$Q_1(n)$	$Q_0(n)$	$Q_2(n+1)$	$Q_1(n+1)$	$Q_0(n+1)$	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	1	1	1	x	0	x	0	1	x
1	1	1	0	0	0	x	1	x	1	x	1

Il ne reste plus qu'à déterminer les fonctions J_i et K_i, par exemple en utilisant des tableaux de Karnaugh. Il faut un tableau pour chacune des entrées J_i et K_i, donc 6 dans notre exemple.

Les entrées de chaque tableau de Karnaugh sont les différents états de ($Q_2(n)$, $Q_1(n)$, $Q_0(n)$).

K_0 :

Q_2Q_1 / Q_0	00	01	11	10
0	x	x	x	x
1	1	1	1	1

$K_0 = 1$

J_0 :

	00	01	11	10
0	1	1	1	1
1	x	x	x	x

$J_0 = 1$

K_1 :

Q_0 \ Q_2Q_1	00	01	11	10
0	x	0	0	x
1	x	1	1	x

$K_1 = Q_0$

J_1 :

Q_0 \ Q_2Q_1	00	01	11	10
0	0	x	x	0
1	1	x	x	1

$J_1 = Q_0$

K_2 :

Q_0 \ Q_2Q_1	00	01	11	10
0	x	x	0	0
1	x	x	1	0

$K_2 = Q_0Q_1$

J_2 :

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	x	x
1	0	1	x	x

$J_2 = Q_0Q_1$

On retrouve bien les résultats précédents.

De la même façon, on peut réaliser n'importe quel compteur ou décompteur, et plus généralement obtenir n'importe quelle séquence d'état (on obtient alors ce que l'on appelle "machine d'état").

Avec des bascules D

La table de transition de la bascule D est :

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

On obtient donc la table des états futurs suivante :

$Q_2(n)$	$Q_1(n)$	$Q_0(n)$	$Q_2(n+1)$	$Q_1(n+1)$	$Q_0(n+1)$	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Les tableaux de Karnaugh correspondants sont :

D_0 :

Q_2Q_1 / Q_0	00	01	11	10
0	1	1	1	1
1	0	0	0	0

D_1 :

Q_2Q_1 / Q_0	00	01	11	10
0	0	1	1	0
1	1	0	0	1

D_2 :

Q_2Q_1 / Q_0	00	01	11	10
0	0	0	1	1
1	0	1	0	1

D'où les équations des entrées des bascules :

$$D_0 = \overline{Q_0} \qquad D_1 = \overline{Q_0}.Q_1 + Q_0.\overline{Q_1} = Q_0 \oplus Q_1 \qquad D_2 = Q_2.\overline{Q_0} + Q_2.\overline{Q_1} + \overline{Q_2}.Q_1.Q_0$$

Le schéma se déduit alors directement de ces équations. On retrouve le schéma précédent.

c) Compteurs/décompteurs

Rappelons à nouveau la table de transition d'une bascule D :

Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

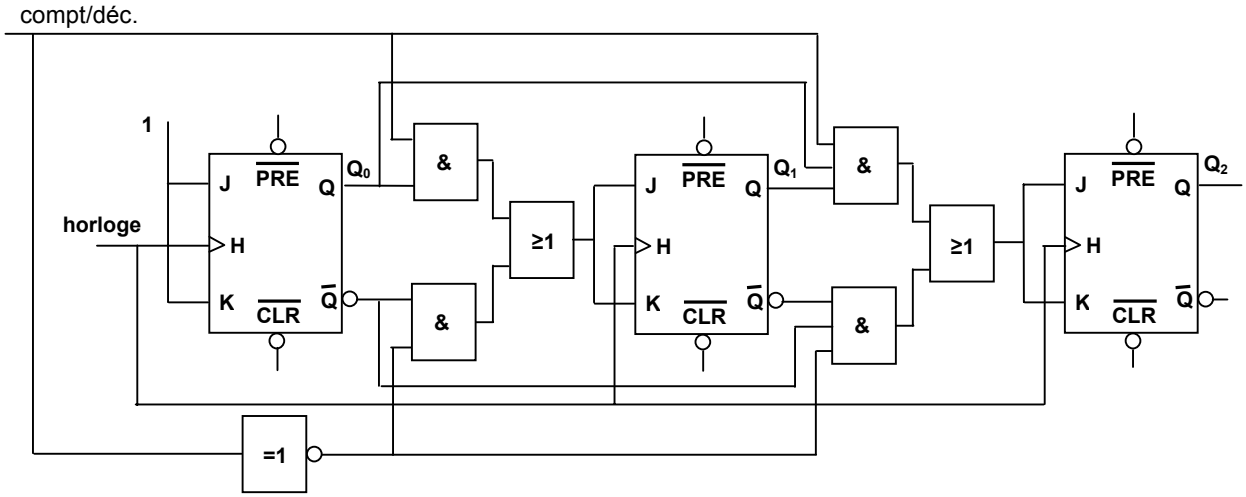
Pour obtenir un décompteur, on établit sa table des états futurs :

$Q_2(n)$	$Q_1(n)$	$Q_0(n)$	$Q_2(n+1)$	$Q_1(n+1)$	$Q_0(n+1)$	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	1	1	1	1	x	1	x	1	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	0	0	1	0	x	x	1	1	x
0	1	1	0	1	0	0	x	x	0	x	1
1	0	0	0	1	1	x	1	1	x	1	x
1	0	1	1	0	0	x	0	0	x	x	1
1	1	0	1	0	1	x	0	x	1	1	x
1	1	1	1	1	0	x	0	x	0	x	1

Si on la compare avec celle du compteur étudiée précédemment, on remarque que si l'on complémente toutes les valeurs des sorties $Q_i(n)$ et celles des $Q_i(n+1)$, et si l'on inverse J et K, on retrouve la table de vérité du compteur. On en déduit donc :

$$J_0 = K_0 = 1$$
$$J_1 = K_1 = \overline{Q_0}$$
$$J_2 = K_2 = \overline{Q_0 . Q_1}$$

Le schéma de ce compteur/décompteur est donc :

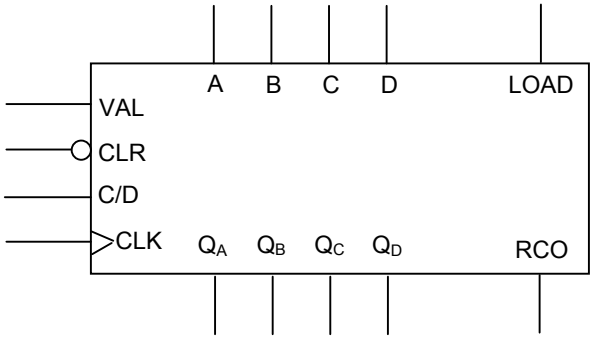


d) Compteurs intégrés

Il existe des circuits intégrés qui permettent de réaliser plusieurs types de compteurs/décompteurs avec un seul circuit.

Le circuit peut alors être vu comme une boite noire qui possède en général les fonctions suivantes :

- une entrée de remise à zéro CLR, asynchrone ou synchrone, généralement active au niveau bas ;
- une entrée de chargement LOAD, en général synchrone, d'une valeur particulière sur les entrées A, B, C, D ;
- une commande de comptage ou de décomptage C/D ;
- une sortie de détection quand toutes les sorties valent 1 pour pouvoir propager une éventuelle retenue RCO (Ripple Carry Output) ;
- une commande de validation ou d'interdiction de comptage VAL.

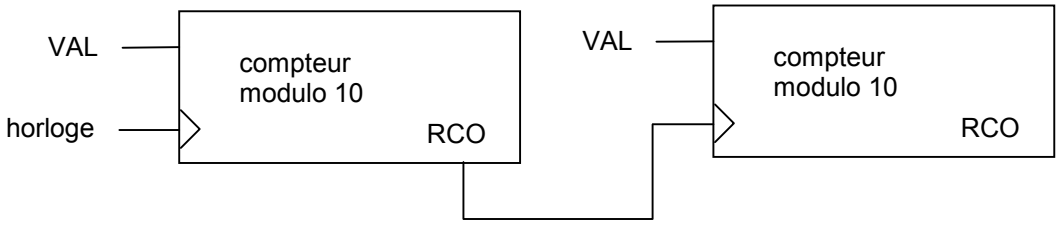


Grâce aux entrées de chargement, on peut obtenir des séquences tronquées.

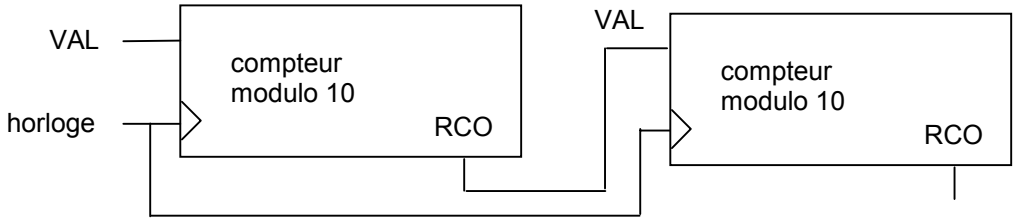
e) Mise en cascade de compteurs

On peut mettre des compteurs en cascade pour obtenir un modulo plus élevé. La sortie du dernier étage pilote une entrée du second compteur.

La mise en cascade peut être asynchrone en utilisant l'entrée horloge du second compteur :



La mise en cascade peut également être synchrone en utilisant l'entrée de validation du second compteur :



La mise en cascade de ces deux compteurs modulo 10 permet de réaliser un compteur modulo 100.

On peut par exemple réaliser un chronomètre en mettant en cascade un compteur modulo 10 et un compteur modulo 6. Le résultat sera donc un modulo 60.

III.3.3) Synchrone vs asynchrone

Les systèmes synchrones possèdent les avantages et inconvénients suivants vis-à vis des systèmes asynchrones :

- leur synthèse est plus simple (on a vu que l'on pouvait synthétiser n'importe quelle séquence, mais comment faire pareil avec un circuit asynchrone ? La réponse sort du cadre de ce cours) ;
- leur consommation est plus importante (car l'horloge fonctionne même quand le système est au repos) ;
- il sont plus lents (car il faut attendre chaque coup d'horloge pour la mise à jour des sorties et/ou états internes) ;
- un système séquentiel peut comporter des états instables, et ceux-ci peuvent durer pendant une période d'horloge (ce qui représente un inconvénient) ;
- …

III.4) Machines d'état

III.4.1) Définitions

a) Machine d'état synchrone

Dans un paragraphe précédent, il a été montré qu'avec des bascules (D, JK, T…), on pouvait réaliser des compteurs synchrones ou asynchrones, mais également des décompteurs.

On peut également réaliser un compteur/décompteur, dont le mode de fonctionnement dépend d'une entrée de commande ("comptage/décomptage").

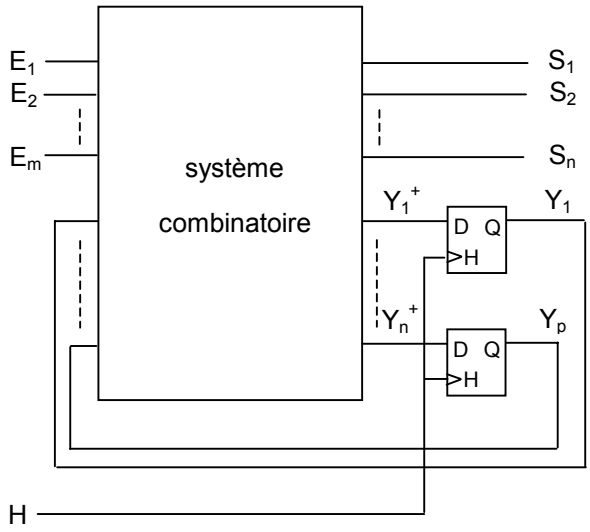
Ces compteurs, décompteurs ou compteurs/décompteurs sont un cas particulier de ce que l'on appelle des machines d'état ("state machine". Cette dernière expression fait référence à la façon dont ces circuits séquentiels sont conçus : la séquence des états est établies, et le circuit permettant d'obtenir cette séquence est synthétisé.

On aurait très bien pu prendre n'importe quelle séquence, à la place de la séquence de comptage ou décomptage. La méthode de synthèse aurait alors été exactement la même, avec les différentes étapes suivantes :

- table de vérité,
- table des états futurs,
- équations logiques (simplifiées) des entrées des bascules (obtenues à l'aide des tables de transition de ces bascules),
- circuit logique.

Dans une machine d'états, on distingue la partie combinatoire et la partie séquentielle. La partie combinatoire correspond à celle qui permet d'obtenir les entrées adéquates pour les bascules dans les compteurs ou décompteurs. La partie séquentielle correspond aux bascules.

Comme pour les compteurs ou décompteurs, il existe les machines d'état synchrones et les machines asynchrones. Pour les machines synchrones, le schéma général est le suivant :



La plupart du temps, ce sont des bascules D qui sont utilisées, synchrones sur front.

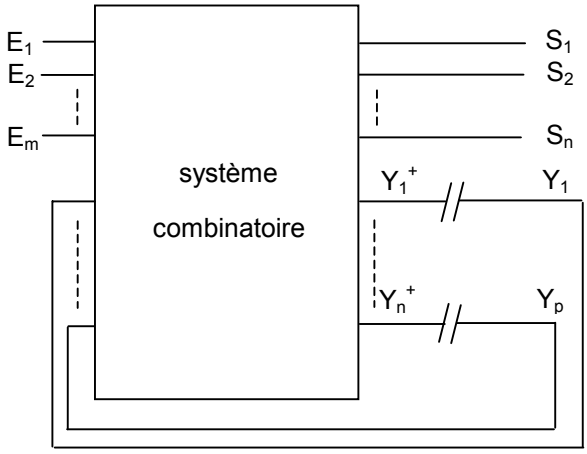
Selon l'application, les variables bouclées Y_i peuvent être utilisées comme des sorties du système, ou comme des variable interne uniquement.

Les entrées des bascules peuvent être considérées comme les états futurs des sorties Y_i car les sorties de ces dernières sont mises à jour à chaque front actif d'horloge ; d'où leur notation Y_i^+.

Un exemple de machine d'état est le compteur/décompteur étudié précédemment, possédant une entrée de commande "comptage/décomptage", et dont les sorties seraient celles des bascules.

b) Machine d'état asynchrone

Dans le cas des systèmes asynchrones, la méthode de synthèse utilise le même principe que pour la méthode synchrone, en considérant que les boucles sont ouvertes non plus par les bascules mais par la pensée. Le schéma devient alors :



La synthèse des systèmes asynchrones est un plus compliquée que celle des systèmes synchrones, même si elle comporte de nombreux points communs avec cette dernière.

III.4.2) Graphes des états ou graphe des transitions

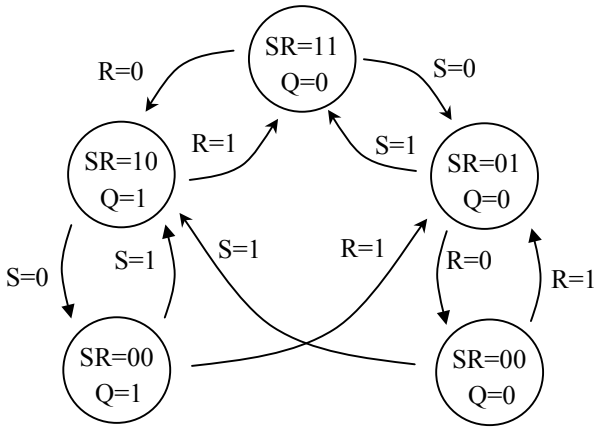
Un graphe d'état représente, sous forme graphique par des cercles, les états stables du système, c'est à dire les états des entrées et les sorties correspondantes.

Le passage d'un état stable à un autre se fait par changement d'une variable d'entrée. Ce changement est matérialisé par une flèche (on dit également "arc orienté") entre un cercle à un autre.

Le système étant séquentiel, un état d'entrée peut correspondre à plusieurs états de sortie. Par exemple, dans une bascule RS, R=S=0 correspond à l'état de mémorisation. Cet état d'entrée peut donc correspondre aux 2 états possible de la sortie S. Le graphe des états de cette bascule comporte donc 2 cercles pour ces 2 états d'entrée.

Voici un rappel de la table de vérité d'une bascule RS, puis le graphe d'états qui en est déduit :

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	non utilisé (0)



Ce mode de représentation permet la synthèse automatique de système séquentiels complexes. Il est utilisé par les méthodes de synthèse modernes.

Partie IV) Technologie des circuits intégrés

Il existe 2 grands types de familles de circuits logiques : la famille TTL et la famille CMOS.

La famille TTL est basée sur l'utilisation de transistors bipolaires (TTL signifie Transistor-Transistor Logic pour désigner le fait que les étages d'entrée et de sortie sont constitués de transistors bipolaires), alors que la famille CMOS (Complementary MOS) est basée sur l'utilisation de transistors à effet de champ MOS (Metal-Oxide Semiconductor), et notamment 2 transistors MOS complémentaires : MOS de type N et MOS de type P.

La technologie TTL est en passe de devenir obsolète. Les composants de cette technologie sont encore fabriqués pour des raisons de maintenance, mais ils vont se raréfier et devenir de plus en plus coûteux.

Cette partie décrit des principes des familles TTL et CMOS, puis leur principales caractéristiques, et enfin leurs performances respectives.

IV.1) Description des familles TTL et CMOS

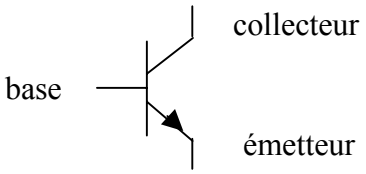
Les circuits de la famille TTL sont basés sur des transistors bipolaires, et les circuits de la famille CMOS sur des transistors à effet de champ.

Dans chacune de ces 2 familles, il existe de nombreuses sous-familles améliorant un ou plusieurs aspect(s) de leur fonctionnement.

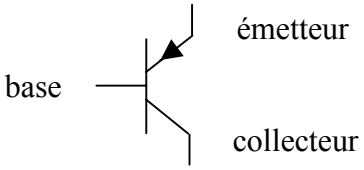
IV.1.1) Technologie utilisée

a) Transistors bipolaires

Les transistors bipolaires peuvent être de type NPN ou PNP. Le transistor NPN est commandé par un courant entrant par la base du transistor ; le transistor PNP est commandé par un courant sortant. Les symboles de ces 2 types sont respectivement :



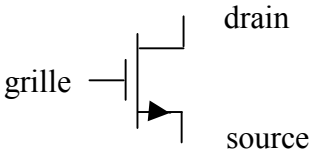
type NPN



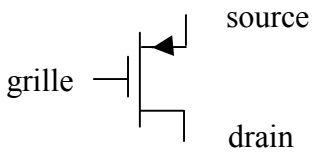
type PNP

b) Transistors MOS

Les transistors MOS peuvent être de type N ou P. Les symboles de ces 2 types sont respectivement :



type N



type P

Le transistor de type N est commandé par une tension grille-source positive ou nulle ; le transistor de type P est commandé par une tension grille-source négative.

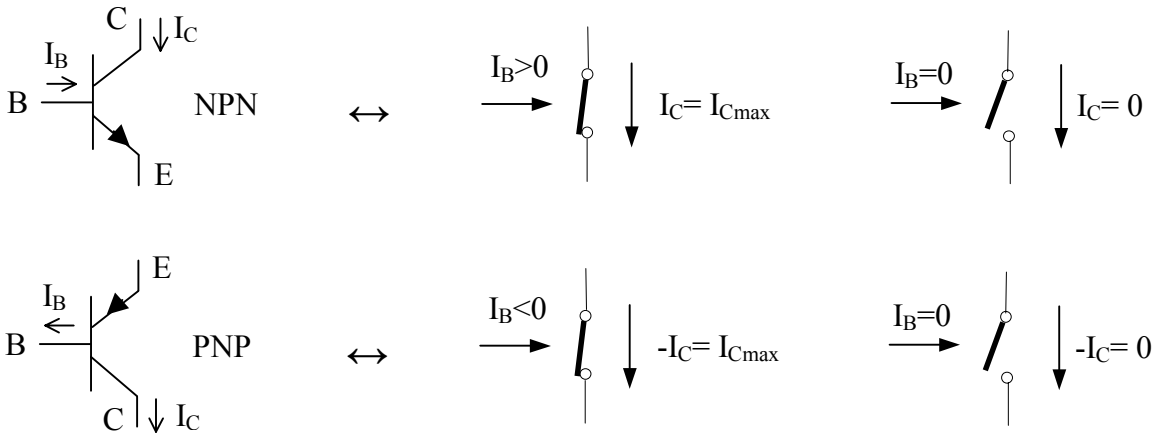
c) Modèles électriques

Un transistor peut être considéré comme un robinet à électrons. Le courant circule dans le sens de la flèche.

Un transistor bipolaire est commandé par un courant, alors qu'un transistor MOS est commandé par une tension.

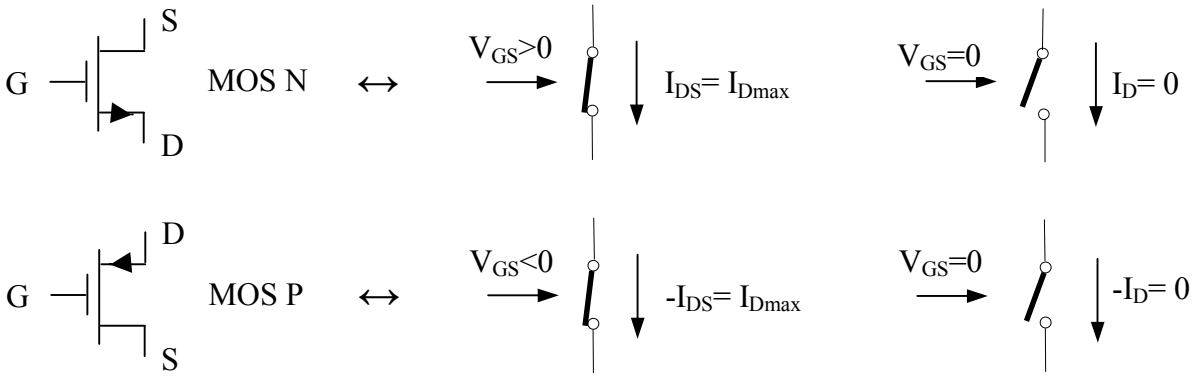
Dans le domaine de la logique, ils sont utilisés en mode tout-ou-rien (encore appelé mode "bloqué-saturé").

Un transistor bipolaire peut être considéré comme un interrupteur commandé en courant :



(en considérant que le sens positif du courant I_C est le sens entrant dans le collecteur).

Un transistor MOS peut être considéré comme un interrupteur commandé en tension :



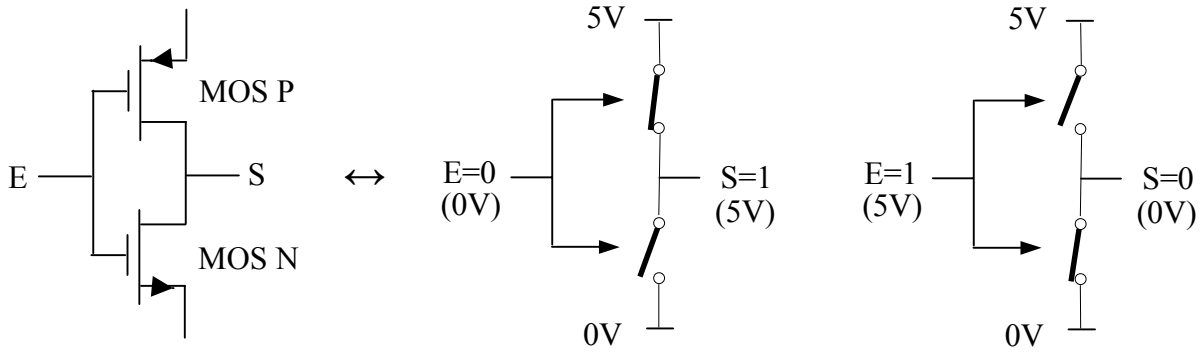
(en considérant que le sens positif du courant I_D est le sens sortant du drain).

d) Réalisation de fonctions logiques élémentaires

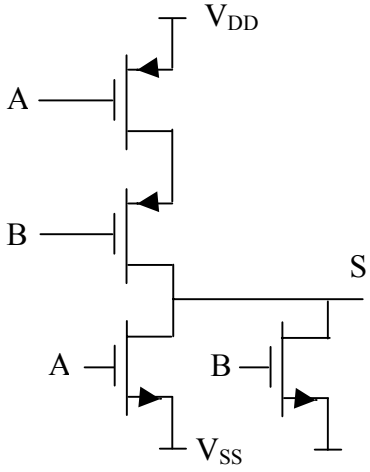
Les fonctions logiques de base sont réalisées par association de plusieurs transistors. Seule la famille CMOS sera abordée dans ce paragraphe.

Fonction NON

La fonction inversion peut être réalisée au moyen de 2 transistors :



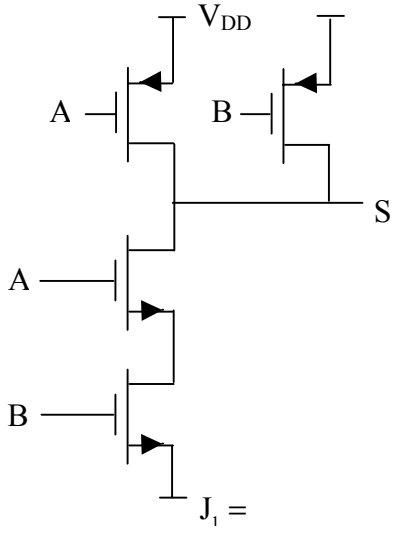
Fonction NON-OU



- Quand A=1 ou B=1, les 2 transistors MOS P sont bloqués. Dans le même temps, l'un des 2 transistors MOS N est saturé, tirant la sortie à 0.
- Quand A=B=0, les 2 MOS P sont saturés, et tirent la sortie à 1.

Remarques :
- La même structure peut être utilisée pour un nombre quelconque d'entrées ;
- En technologie CMOS, l'alimentation est généralement désignée par V_{DD}, et la masse par V_{SS}.

Fonction NON-ET



- Quand A=0 ou B=0, les 2 transistors MOS N sont bloqués. Dans le même temps, l'un des 2 transistors MOS P est saturé, tirant la sortie à 1.
- Quand A=B=1, les 2 MOS N sont saturés, et tirent la sortie à 0.

Remarque : comme pour le NON-OU, la même structure peut être utilisée pour un nombre quelconque d'entrées.

Autres fonctions

La technologie CMOS permet également de réaliser des combinaisons de portes, directement, c'est à dire avec moins de transistors que par l'association de plusieurs portes logiques.

IV.1.2) Sous-familles

a) Famille TTL

En plus de la technologie TTL standard existent d'autres sous-familles, et notamment la famille à diodes Schottky : il s'agit de diodes à commutation rapide, qui remplacent les transistors de l'étage d'entrée. On trouve les séries :

- 74L (Low Power) : elle comporte des résistances internes plus grandes, ce qui réduit sa consommation mais augmente les temps de propagation ;
- 74H (High Speed) : ses résistances internes sont plus faibles ; sa consommation est plus importante mais les temps de propagation sont plus faibles ;
- 74S (Schottky) : basée sur l'utilisation de diode de commutation rapides en entrée, elle est 2 fois plus rapide que la série standard ; ses résistances internes sont 2 fois plus faibles. On trouve également la série 74ALS (Advanced Low Power Shottky), qui est la plus rapide et dont la consommation est la plus faible.

Les circuits intégrés TTL sont référencés par :

74 *xx nnn*

où *xx* est la technologie (1 ou plusieurs caractères : L, LS, ALS, ...), et *nnn* le numéro de la fonction logique réalisée (2 à 4 chiffres : 00 à plus de 1000).

b) Famille CMOS

La série standard est la série 4000. Ces composants sont référencés :

4*nnn x*B,

où *nnn* est le numéro de la fonction logique réalisée (3 ou 4 chiffres à partir de 000), et *xB* est le type d'étage de sortie : B = bufferisé, UB = non bufferisé.

Il existe également des sous-familles dont les référence commencent par 74 pour signaler le fait que le brochage des circuits intégrés est compatible avec ceux de la série TTL :

- 74HC / 74HCT / 74HCU, technologie Hi-Speed CMOS :
  elle combine les avantages de faible consommation de la famille CMOS 4000 avec la haute vitesse et la capacité de commande de la famille TTL-LS ;
  - 74HC*xxxx* : alimentation 2 à 6V ; consommation statique négligeable ; haute immunité au bruit ;
  - 74HCT*xxxx* : idem 74HC mais avec compatibilité avec la famille TTL (mêmes seuils de commutation) ;

  Référence :

  74 HC *nnn*

  où *nnn* est une fonction logique correspondante de la famille TTL, ou :

  74 HC 4*nnn*

  où *nnn* est une fonction logique de la famille CMOS standard (série 4000).

- 74LV, technologie LV-HCMOS
  Elle s'alimente en faibles tensions (Low Voltage) : de 1,2 à 3,6 V, typiquement 3,3V. Elle est indiquée lorsque la vitesse de la 74HC/HCT est suffisante et qu'il est requis une baisse importante de la consommation. Elle est typiquement utilisée dans les systèmes portables.

IV.2) Caractéristiques des familles TTL et CMOS

Les principales caractéristiques des circuits intégrés des familles TTL et CMOS et des portes logiques qu'ils comportent, sont :

- les tensions d'alimentation des circuits intégrés,
- les tensions d'entrée et de sortie correspondant aux niveaux logiques 0 ou 1,
- la consommation statique (au repos) et la consommation dynamique (en fonctionnement),
- la sortance : nombre maximal de portes logiques pouvant être connectées en sortie d'une porte donnée pour une utilisation normale,
- le temps de propagation : durée nécessaire pour que la fonction logique d'une porte soit disponible sur sa sortie, lorsqu'une nouvelle entrée lui est appliquée,
- l'immunité au bruit,
- la manière dont il faut gérer des entrées non-utilisées,
- et d'autres considérations…

IV.2.1) Alimentation

TTL

Les circuits logiques TTL doivent être alimentés sous 5V +/- 5%, soit entre 4,75V et 5,25V. Si cette tension est dépassée, il y a risque de claquage de la jonction base-émetteur du transistor d'entrée.

CMOS

Les circuits logiques CMOS s'alimentent sous des tensions différentes selon la sous-famille :

- série standard : entre 3V et 18V ;
- sous-famille 74HC : entre 2 et 6V ;
- sous-famille 74LV : entre 1,2V et 3,6V.

IV.2.2) Niveaux de tension de courant

a) Niveaux de tension d'entrée

Lorsque l'on applique une tension en entrée d'une porte logique, elle est interprétée comme un niveau 0 ou un niveau 1. Ces 2 niveaux logiques correspondent à des plages de tension, garanties par le fabricant :

- entre 0V et V_{iLmax}, la tension d'entrée sera considérée comme un niveau 0 ;

- entre V_{iHmin} et V_{cc} (tension d'alimentation du circuit intégré), la tension d'entrée sera considérée comme un niveau 1.

Entre ces 2 seuils, le fonctionnement du circuit pourra être aléatoire.

TTL

V_{iLmax}=0,8V ; V_{iHmin}=2V

CMOS

V_{iLmax}=30% V_{DD} ; V_{iHmin}=70% V_{DD}

b) Niveaux de tension de sortie

De le même manière que pour les tensions d'entrée, les niveaux logiques de sortie correspondent à des plages de tension, garanties par le fabricant :

- le niveau 0 de sortie correspondra à une tension comprise entre 0V et V_{oLmax} ;
- le niveau 1 de sortie correspondra à une tension comprise entre V_{oHmin} et V_{cc} ;

TLL

V_{oLmax}=0,4V ; V_{oHmin}=2,4V

CMOS

V_{oLmax}=0,1V ; V_{oHmin}= V_{DD} - 0,1V

On constate que dans la technologie CMOS, les niveaux de sortie sont très proches des potentiels d'alimentation.

Le seuil de commutation de l'étage d'entrée d'une porte CMOS est d'environ 50% de la tension d'alimentation et la courbe de transfert en tension (tension de sortie en fonction de la tension d'entrée) est quasiment idéale.

IV.2.3) Consommation

Il faut distinguer la consommation statique, qui est la consommation du circuit intégré quand il est alimenté mais qu'aucun niveau d'entrée ne change, de la consommation dynamique dans le cas contraire.

En multipliant le courant circulant dans le circuit par la tension d'alimentation, on obtient la consommation en puissance.

a) Consommation statique

TTL

Dans le cas des circuits TTL, avec V_{cc}=5V :

10mW

CMOS

Dans le cas des circuits CMOS, la consommation statique est nulle. Par exemple, pour l'opérateur NON-ET, alimenté avec V_{DD}=5V :

2,5nW

b) Consommation dynamique

TTL

Dans le cas des circuits TTL, l'augmentation de la consommation en régime dynamique, par rapport à celle du régime statique, est **pratiquement négligeable**.

CMOS

Dans le cas d'une porte logique CMOS, la consommation augmente avec la fréquence du signal d'entrée. Un transistor MOS se comporte comme une capacité de 5pF en parallèle avec une résistance très élevée ($>10^{10}\Omega$), vue de son entrée (sa grille). Le courant de sortie augmente proportionnellement à la fréquence du signal d'entrée (d'où la puissance des blocs d'alimentations des ordinateurs). Par exemple pour l'opérateur NON :

0,1 mW à 100 kHz,
1 mW à 1 MHz.

IV.2.4) Sortance

La sortance est le nombre maximal de portes logiques (on parle également de "charges logiques") pouvant être connectées en sortie d'une porte donnée, en régime dynamique (c'est à dire quand les niveaux logiques varient au cours du temps).

La sortance à l'état haut, c'est à dire le nombre maximal de portes pouvant être commandées par cette porte pour leur imposer un niveau haut en entrée peut être différente de la sortance à l'état bas, la caractéristique équivalente pour le niveau bas.

Si la sortance est dépassée, le fonctionnement du circuit logique peut devenir incorrect et ne plus correspondre au fonctionnement théorique.

TTL

La sortance d'un opérateur TTL, en régime dynamique, est de l'ordre de quelques unités à quelques dizaines, selon la sous-famille utilisée.

CMOS

Le courant d'entrée d'un transistor à effet de champ étant très faible (en général négligeable), ça n'est pas cette caractéristique qui limite la sortance d'une porte logique de type CMOS. Ce qui limite cette sortance, c'est le fait qu'une charge logique se comporte de son entrée comme une capacité (voir plus haut). Plus la fréquence d'utilisation est élevée, plus la sortance est petite. Par exemple, à 1Mhz, elle est égale à 50 environ.

IV.2.5) Temps de propagation

Le temps de propagation d'une porte logique est le temps que met un signal logique pour la traverser.

Cette durée est différente selon que le niveau de sortie passe du niveau 0 au niveau 1 ou du niveau 1 au niveau 0. Le temps de propagation est la moyenne de ces durées.

Si on note t_{pHL} le temps de passage du niveau 1 au niveau 0 en sortie (High to Low) et t_{pLH} le temps de passage du niveau 0 au niveau 1, le temps de propagation est défini par :

$$t_p = \frac{t_{pHL} + t_{pLH}}{2}$$

Le temps de propagation varie avec la charge. Les valeurs ci-dessous sont définies pour une charge purement capacitive de 15pF.

TTL

Dans le cas de la famille TTL, les valeurs moyennes sont les suivantes :

t_{pHL}=7ns (max. : 15ns) t_{pLH}=11ns (max. : 22ns)

d'où un temps de propagation moyen

$$t_p=9ns$$

CMOS

Dans le cas de la famille CMOS standard, le temps de propagation moyen est :

$$t_p=35ns$$

pour une tension d'alimentation de 5V. Pour une alimentation de 10V, ce temps descend à 25ns, mais on général on cherche plutôt à diminuer la tension d'alimentation, pour des raisons de consommation. Les opérateurs CMOS standard sont donc moins rapides que ceux de la famille TTL, mais ceux de la série CMOS rapide 74HC sont du même ordre (une dizaine de ns).

IV.2.6) Immunité aux bruits

Toute variation parasite d'un signal constitue un bruit. Ce bruit peut entraîner des commutations intempestives des opérateurs logiques. L'immunité au bruit est la variation du signal d'entrée n'entraînant pas de modification de la sortie. Il s'agit d'une marge de sensibilité au bruit.

Au niveau haut, elle est égale à :

$$V_{oHmin}-V_{iHmin}$$

et à l'état bas :

$$V_{iLmax}-V_{oLmax}$$

Remarque : contrairement au domaine analogique, où tout bruit en entrée d'un circuit ou interne, aura des conséquences sur le signal de sortie, ça n'est pas le cas dans un circuit numérique : si l'amplitude du bruit est inférieure à la marge de sensibilité, celui-ci n'aura aucune conséquence sur le fonctionnement du système. Cette propriété est un des principaux avantages du domaine numérique sur le domaine analogique.

TTL

$$V_{oHmin}-V_{iHmin}=2,4-2=0,4V$$
$$V_{iLmax}-V_{oLmax}=0,8-0,4=0,4V$$

CMOS

$$V_{oHmin} - V_{iHmin} = V_{DD} - 70\% V_{DD} = 30\% V_{DD}$$
$$V_{iLmax} - V_{oLmax} = 30\% V_{DD} - 0 = 30\% V_{DD}$$

Pour V_{DD}=5V, la marge d'immunité au bruit est de 1,5V, alors qu'elle n'est que de 0,4V pour la famille TTL.

IV.2.7) Entrées non-utilisées

Dans le cas des circuits TTL, les entrées inutilisées peuvent rester en l'air (non-raccordées). Par contre, dans les circuits CMOS, elles doivent obligatoirement être raccordées à 0V ou V_{DD}.

TTL

Dans le cas d'une porte TTL, une entrée en l'air correspondra au niveau 1. Néanmoins, il vaut mieux raccorder toute entrée non utilisée :

- à la masse pour la mette à 0 ;
- à Vcc, par l'intermédiaire d'une résistance supérieure à 1kΩ pour la mettre à 1.

CMOS

Dans le cas des circuits CMOS, une entrée inutilisée va prendre un état aléatoire.

Ceci est dû au fait que le courant d'entrée des transistors MOS est extrêmement faible : selon que les charges électrostatiques ambiantes s'accumulent ou non dans les connexions d'entrée, le niveau d'entrée peut correspondre à un niveau 0 ou à un niveau 1.

Il faut donc la relier :

- à la masse pour la mette à 0 ;
- à Vcc pour la mettre à 1.

Cette grand sensibilité aux charges électrostatiques fait que les circuits intégrés CMOS peuvent être détériorés par simple contact avec un corps chargé, comme par exemple le corps humain. Il faut penser à se "décharger" avant toute manipulation d'un circuit CMOS ou plus généralement d'une carte électronique (qui a de grandes chances de comporter des circuits CMOS), par exemple en touchant une prise de terre ou un objet métallique relié à la terre.

De plus, pour limiter l'influence des perturbations électriques dans un montage à base de circuits CMOS, on insère en général un condensateur de 10 à 100 nF entre les bornes d'alimentation du circuit intégré, le plus près possible de celles-ci.

IV.3) Association de portes des familles TTL et CMOS

On peut utiliser des circuits TTL et des circuits CMOS dans un même montage, mais il faut veiller à respecter certaines règles.

IV.3.1) TTL vers CMOS

On considère le cas où la sortie d'une porte TTL doit être appliquée en entrée d'une porte CMOS.

On a vu que la tension minimale d'entrée d'une porte CMOS pour être considérée comme un niveau 1 était :

$$V_{iHmin}=3,5V$$

De même, la tension maximale de sortie d'une porte TTL correspondant à un niveau 1 est égale à :

$$V_{oHmin}=2,4V$$

On voit bien que cela pose un problème : un niveau 1 à la sortie d'une porte TTL risque de ne pas être "vu" par la porte CMOS.

Pour remédier cela, il faut ajouter une résistance entre la sortie de la porte TTL (de l'ordre du kΩ) et V_{cc}, ce qui a pour effet de "tirer" le potentiel correspondant au niveau 1 vers V_{cc}.

IV.3.2) CMOS vers TTL

On considère le cas où la sortie d'une porte CMOS doit être appliquée en entrée d'une porte TTL.

Lorsque la sortie de la porte CMOS est à l'état bas, le courant généré par la porte TTL (et sortant par son entrée est relativement important (jusqu'à 1,6mA). Ce courant est trop important pour une porte CMOS standard.

On doit dont utiliser un circuit spécial, appelé "circuit tampon" (par exemple de circuit de référence 4050), pour réaliser l'interface CMOS standard vers TTL. Par contre, les circuits de la série CMOS 74HC peuvent piloter directement des portes TTL.

IV.4) Avantages et inconvénients des circuits des familles TTL et CMOS

La famille CMOS standard est plus lente (voir temps de propagation) que la famille TTL, ce qui représente son principal inconvénient. Mais avec les séries rapides 74HC, les temps de propagation des circuits des 2 familles sont du même ordre.

Du point de vu de la consommation, la différence entre les circuits TTL et CMOS se fait sentir surtout en régime statique : on a vu que celle de la famille CMOS était quasi-nulle.

Un autre avantage de la famille CMOS sur la famille TTL est sa possibilité de grande intégration (LSI : Large Scale Integration). Cet avantage est décisif pour la miniaturisation des ordinateurs et autres systèmes numériques actuels.