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

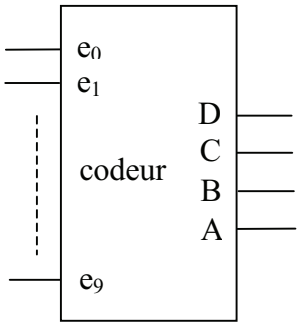
Un codeur (sous-entendu binaire) est un circuit logique codant en sortie l'indice de son entrée active (= "à 1", en logique positive). Sa condition de fonctionnement est donc qu'il n'y ait qu'une seule entrée active en même temps.

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.

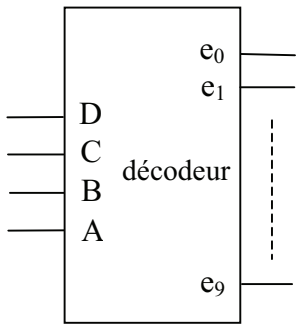
On peut donc déterminer directement la fonction à partir de la table de vérité, et ceci quel que soit le nombre d'entrées.

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

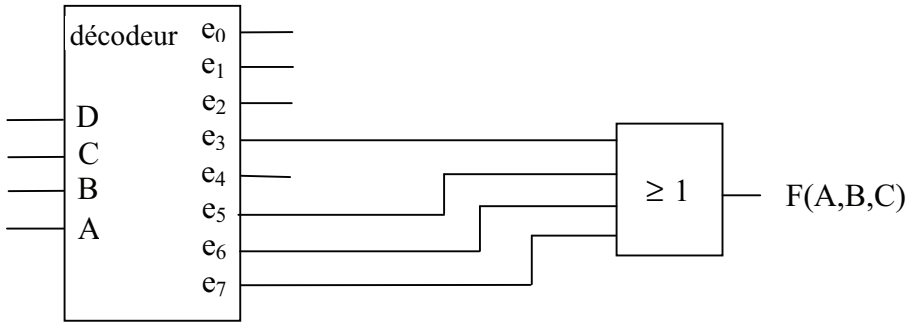
On cherche à réaliser la fonction f (A, B, C) = Σ (3, 5, 6, 7) avec un décodeur et une porte logique. On remplit la table de vérité de manière adéquate :

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.

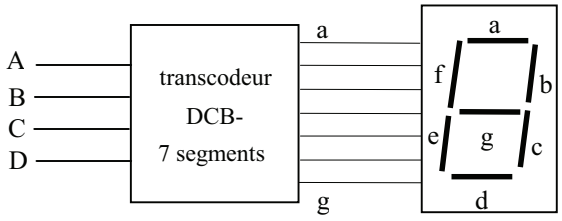
II.2.1) Transcodeur DCB-7 segments

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

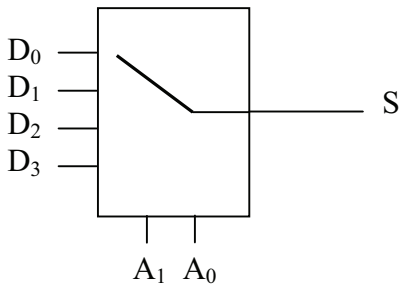
Principe

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 :

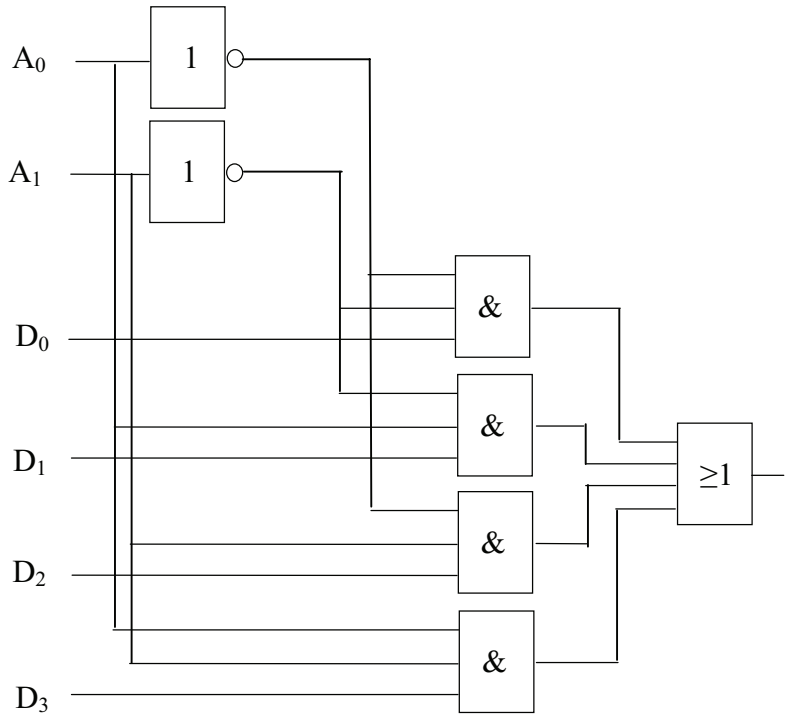
A_1	A_0	Entrée sélectionnée
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

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 :

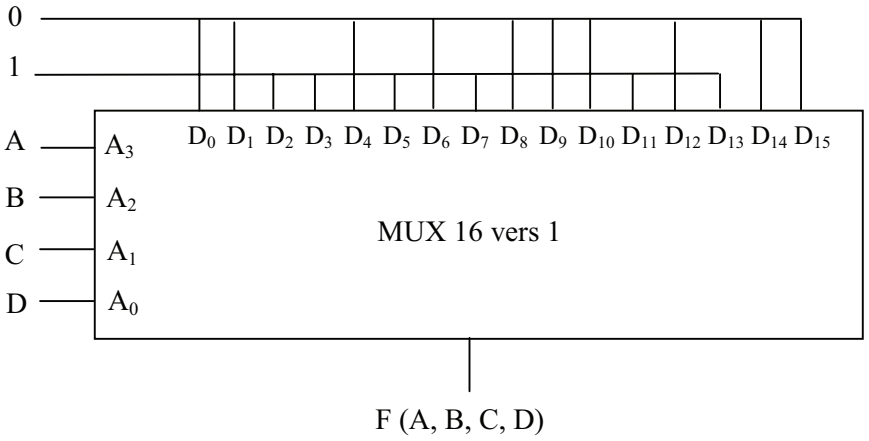


Exemple d'application d'un multiplexeur : réalisation d'une fonction quelconque

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=$\overline{D}$.

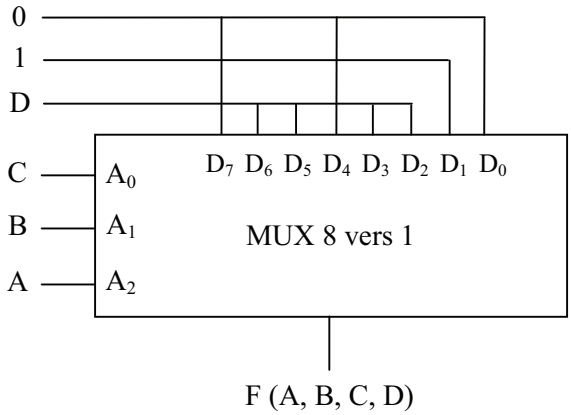
Par exemple, la première adresse : (A, B, C)=(0, 0, 0) correspond aux 2 première lignes de la table de vérité. Il faut trouver l'expression de F sur ces 2 lignes. On constate que l'on a F=0. Il faut donc relier l'entrée D_0 du multiplexeur (celle qui est sélectionnée par l'adresse (0,0,0)) à 0 :

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.

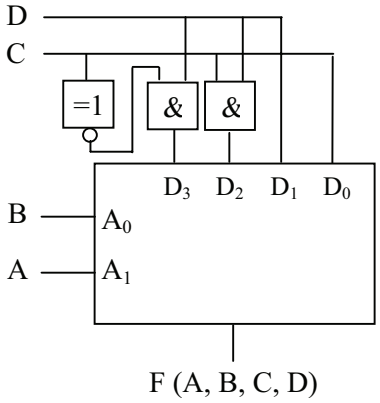
On choisit d'utiliser les variables A et B comme entrées d'adresse. Par exemple, pour la première adresse (0,0), on détermine l'expression de F en fonction de C et D (les 2 variables non-utilisées comme entrées d'adresse), sur les 4 premières lignes de la table de vérité :

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 :

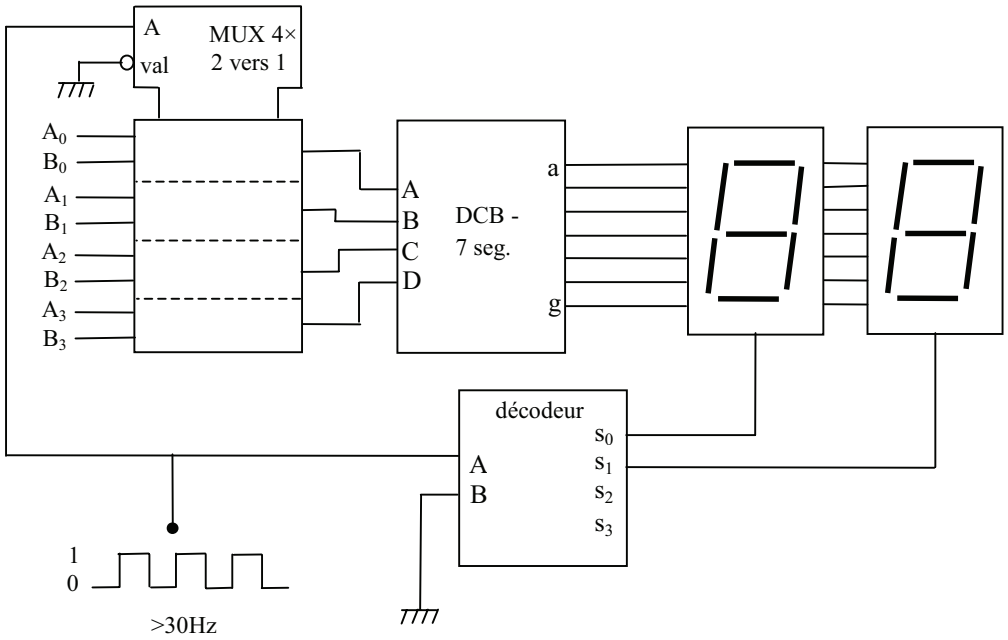


Autre exemple d'application : commande de plusieurs afficheurs 7 segments

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 :



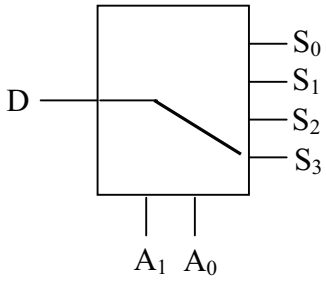
II.3.2) Démultiplexeurs

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 $a_0<b_0$.

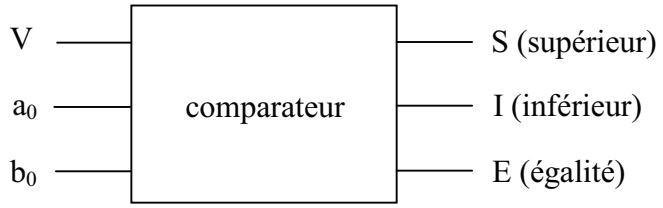
La table de vérité de ce comparateur est la suivante :

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.

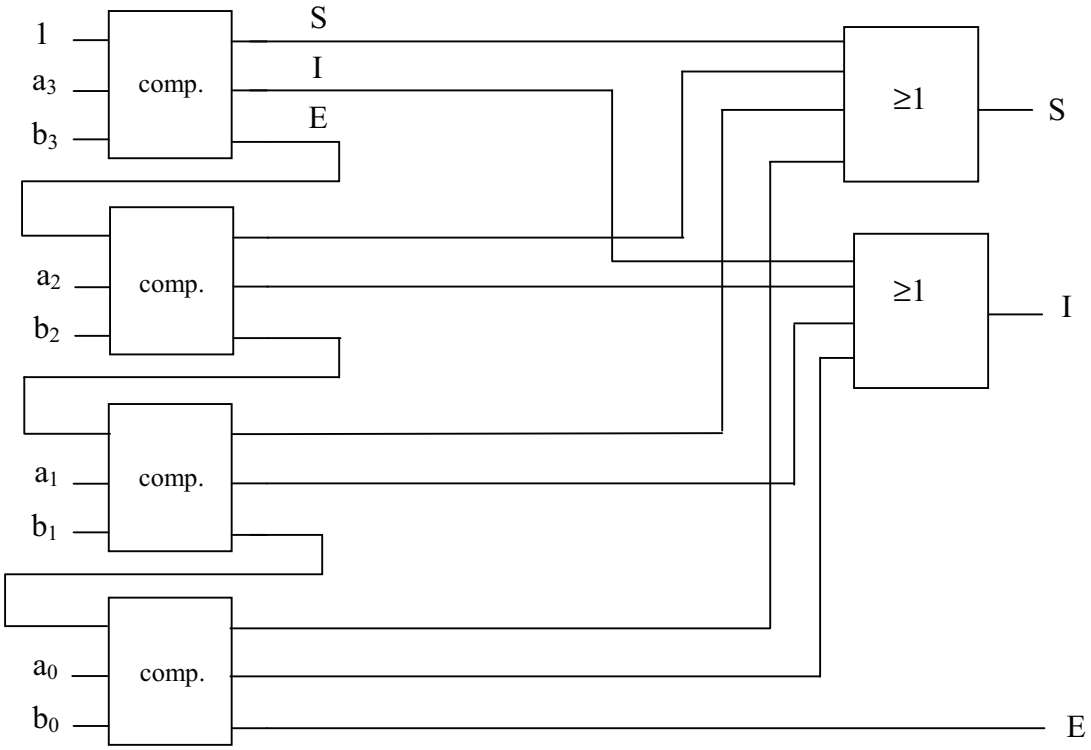
Si le bit de poids le plus fort du 1er mot est 1 et celui du 2^{e} sont les mêmes, on n'est pas renseigné sur la comparaison de ces 2 mots donc il faut aller voir le bit de poids immédiatement inférieur. Et ainsi de suite jusqu'aux bits de poids le plus faible.

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

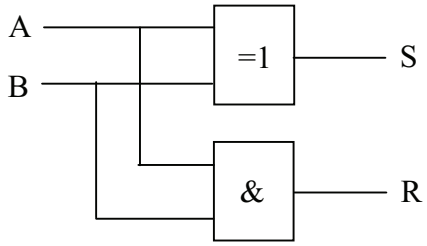
La table de vérité d'un additionneur de deux bits A et B comporte 2 sorties : la sortie S et la retenue R :

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

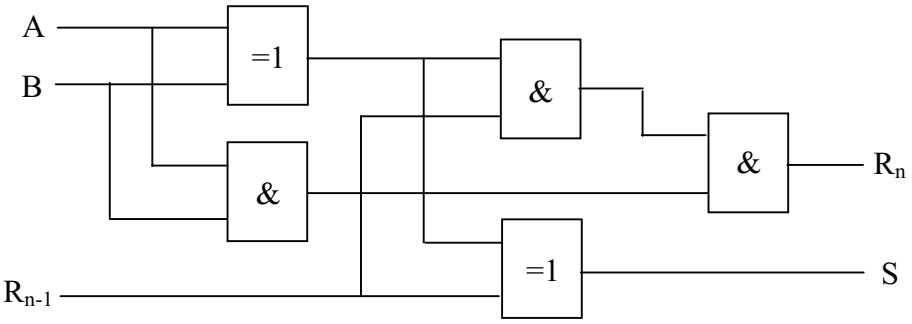
L'expression de la sortie somme S est :

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

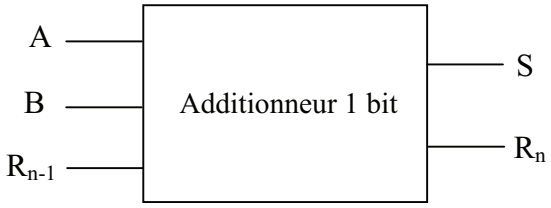
L'expression de la retenue de sortie R_n est :

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

Le schéma correspondant est :

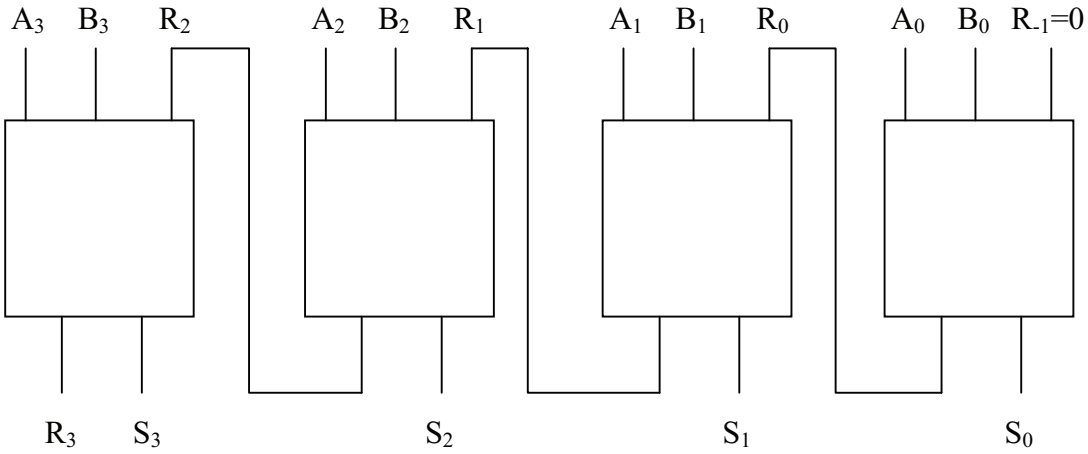


On peut représenter ce circuit sous la forme d'une boite noire :



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 :

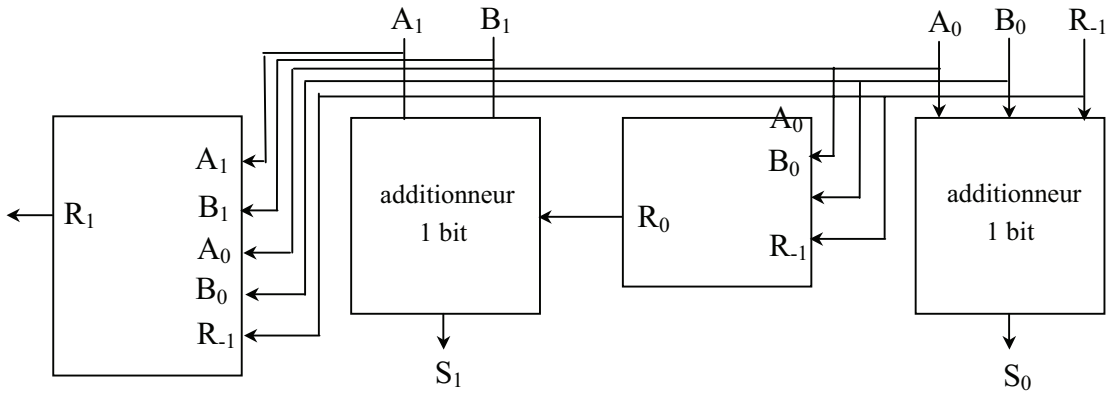
$$S_n = R_{n-1} \oplus (A_n \oplus B_n)$$

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

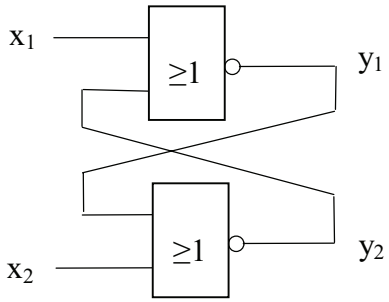
La bascule RS est la plus simple de toutes les bascules. Elle est à la base des autres : la bascule JK et la bascule D, entre autres.

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) :

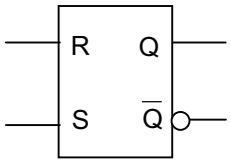
R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	non utilisé (0)

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

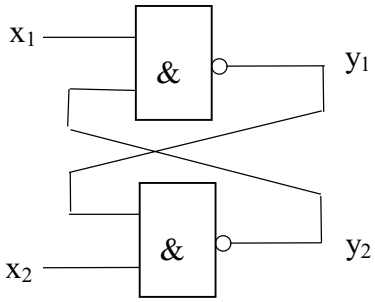
d'où

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

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

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

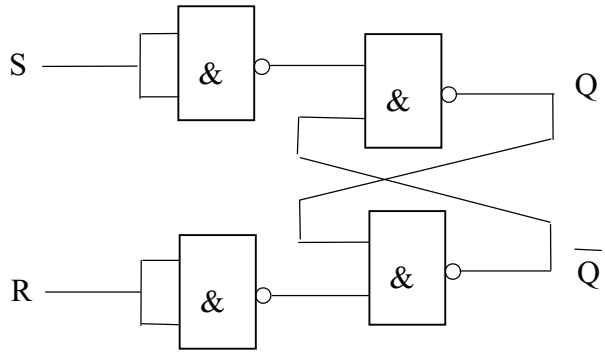
On voit que l'entrée x_1 n'a plus un rôle de mise à 0 mais un rôle de mise à 1 et x_2 un rôle de mise à 0. Pour que cette nouvelle table de vérité corresponde à celle de la bascule RS à portes NOR, il faut donc d'abord complémenter ces 2 entrées, ce qui donne :

$\overline{x_1}$	$\overline{x_2}$	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

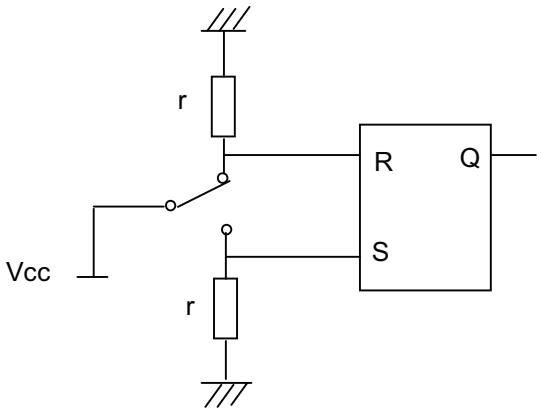
Les 2 bascules RS étudiées ci-dessus sont asynchrones, c'est à dire qu'il n'existe pas de signal par rapport auquel la mise à jour des sorties serait synchronisée ; dès qu'un changement apparaît en entrée, les sorties sont mises à jour immédiatement (plus exactement, après une durée très faible correspondant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



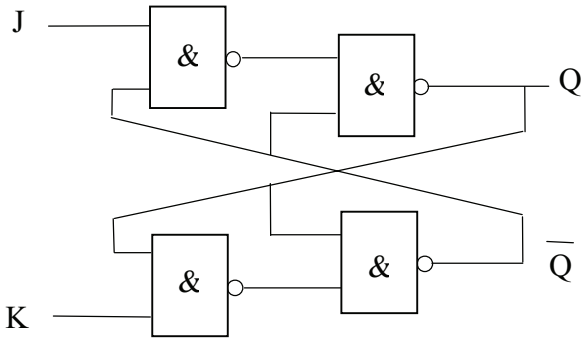
Quand le commutateur est en position haute, R=1 et S=0 donc Q=0. Quand il change de position, on passe par l'état R=S=0, donc on est dans l'état de mémorisation et les sorties ne changent pas. Quand il est en position basse, on a R=0 et S=1, donc Q=1.

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

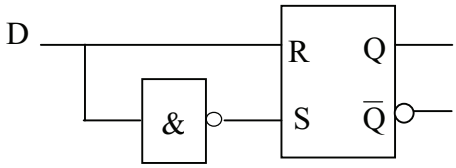
Son équation logique est donc :

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

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

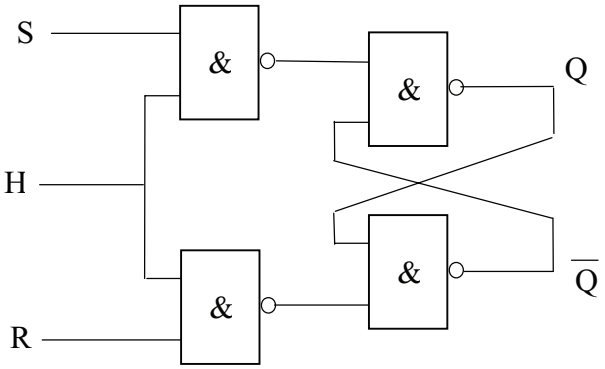
La synchronisation des circuits séquentiels peut se faire de trois façons différentes :

- 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 :

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

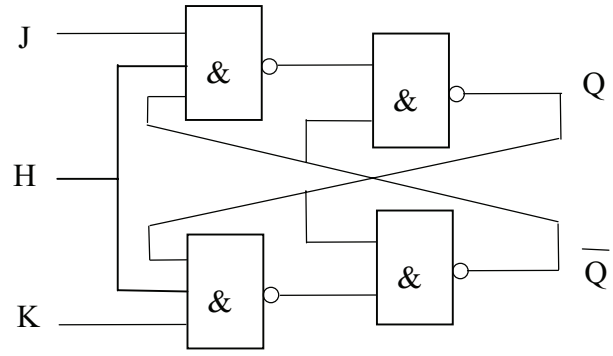
et de remplacer R par R.H et S pas S.H :

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

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).

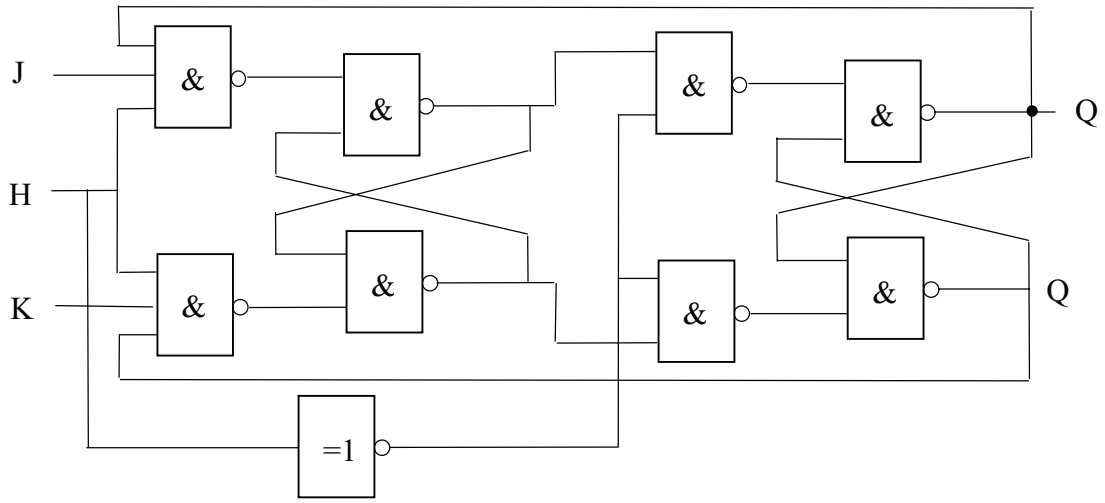
Son équation logique est :

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

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 :

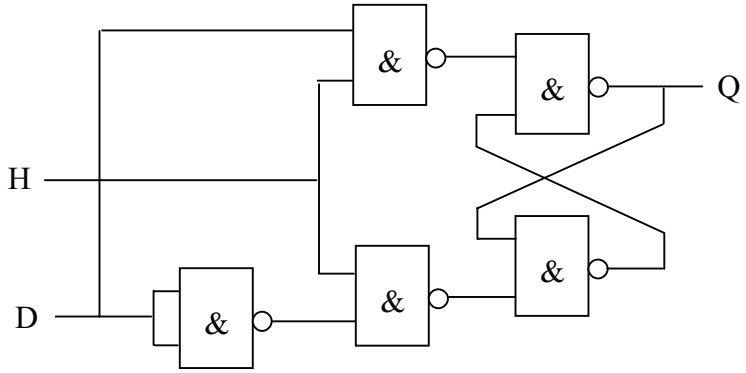


La bascule maître-esclave est donc une solution à l'oscillation existant pour la combinaison J=K=1. Une autre solution consiste à utiliser un déclenchement sur front (voir bascules synchrones, plus loin).

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.

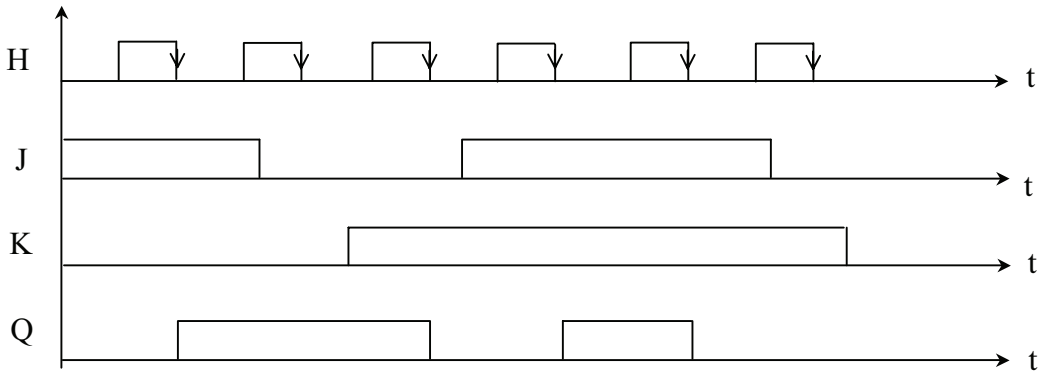
Bascule J K à déclenchement sur front (edge triggered)

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)

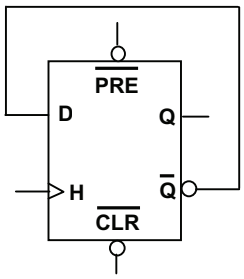
Dans une bascule D à déclenchement sur front, l'entrée D n'est prise en compte qu'au front montant (ou descendant) de l'horloge.

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

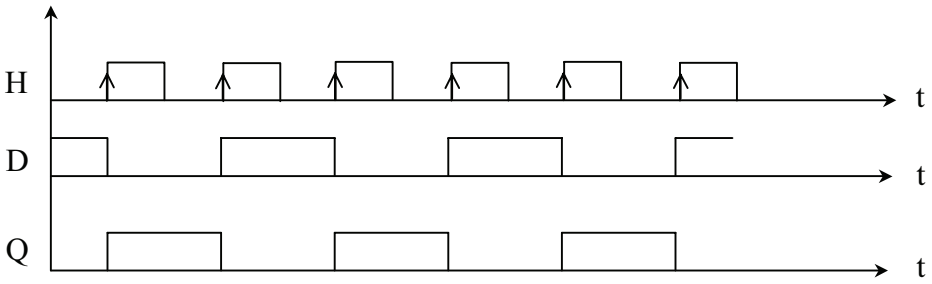
Quand il y a un front actif la sortie recopie l'entrée sinon la sortie garde son état précédent.

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.

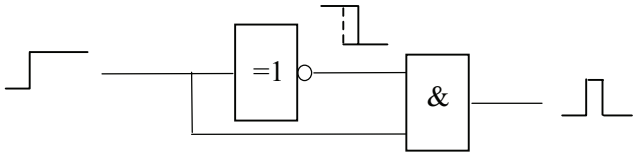
On constate que la sortie Q possède une fréquence 2 fois plus petite que celle de l'horloge. On a donc réalisé un diviseur de fréquence par 2.

Cette propriété est à la base des compteurs.

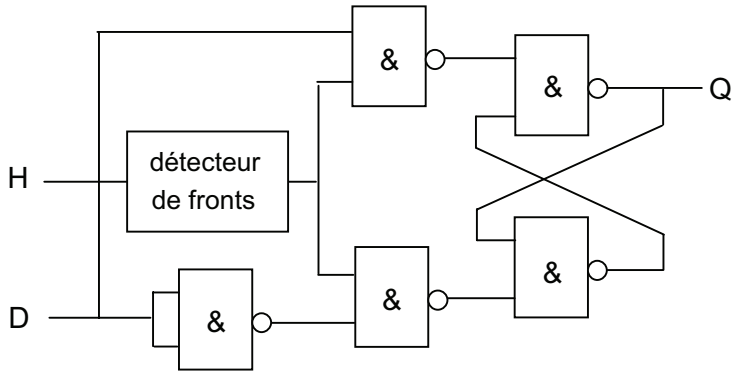
De même, si l'on place les entrée J et K d'une bascule JK à 1, la sortie changera d'état à chaque front actif d'horloge. Le signal en sortie aura une fréquence divisée par deux par rapport à celle de l'horloge.

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

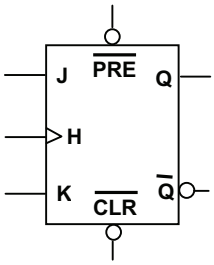
Les entrées de bascules que nous avons étudiées jusqu'ici sont appelées entrées synchrones car les changements des sorties qu'elles provoquent sont synchronisés sur un signal d'horloge H.

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 :

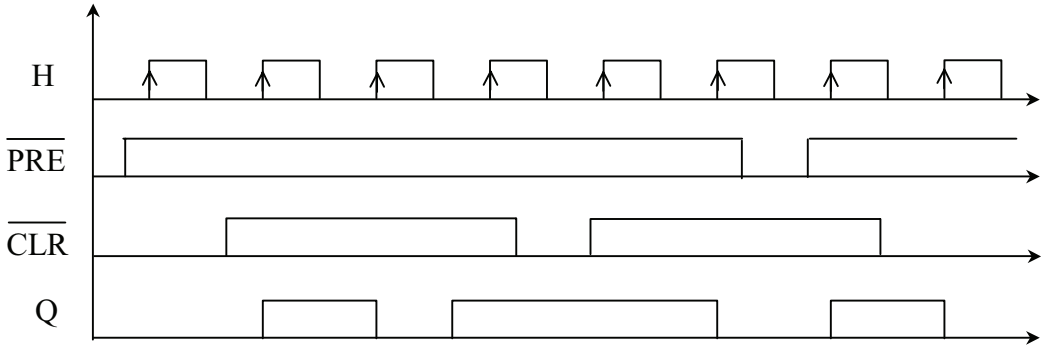
- mise à 1 : SET ou PRESET ou RAU (Remise A Un) ;
- mise à 0 : RESET ou CLEAR ou RAZ (Remise A Zéro).

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



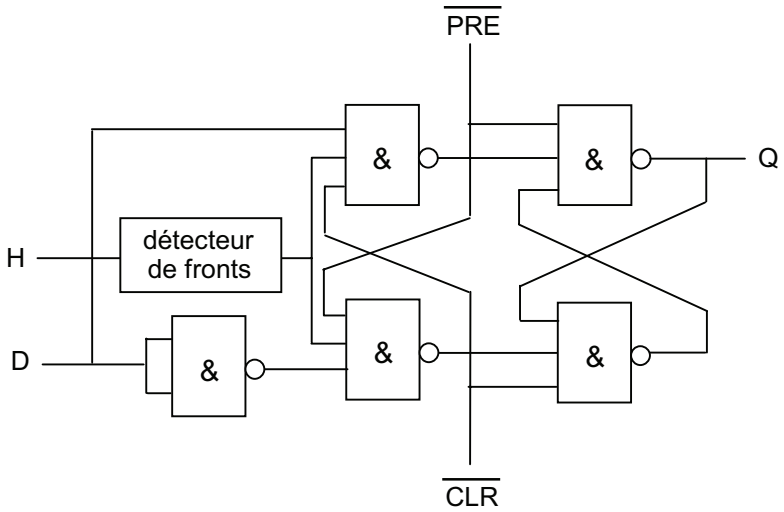
Elles sont généralement actives à l'état bas. Par exemple, un 0 sur l'entrée CLR provoque une remise à zéro de la bascule quelles que soient les valeurs des entrées synchrones. Ces deux entrées asynchrones ne peuvent être actives (=0) en même temps.

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.