Chapitre 13
Circuits logiques

Afin de compléter tout ce qui a été vu jusqu'à présent et de vous préparer à une étude un peu plus profonde du matériel composant un ordinateur, nous présentons ici une introduction aux circuits logiques, qui sont les éléments de base d'un ordinateur.

13.1 Algèbre de Boole

Les circuits logiques des ordinateurs et autres systèmes numériques sont conçus, et leur comportement analysé au moyen de ce qu'on appelle l'algèbre de Boole[1]. L'algèbre de Boole, comme toute autre algèbre, comprend un ensemble d'éléments E (variables et constantes), un ensemble de fonctions F qui opèrent sur les membres de E, et un ensemble de lois de base, appelées *axiomes*, qui définissent les propriétés de E et de F. Les variables, dans une algèbre de Boole, ont la valeur 0 ou 1; si une telle algèbre possède n variables, il existe un ensemble de 2^n permutations de ces variables.

Dans cette algèbre, il n'existe que trois opérations : OU logique (représenté par le signe +), ET logique (représenté par le signe · ou une simple concaténation), et NON logique (représenté par le signe '). Les deux premiers signes sont parfois représentés différemment, mais le plus et le point sont aussi utilisés dans d'autres algèbres; notez qu'il n'y a rien qui corresponde à la multiplication ou à la division, dans l'algèbre de Boole. Le troisième opérateur, celui de négation, est parfois représenté par une barre située au dessus du symbole; pour des raisons de commodité, nous utiliserons l'apostrophe (prime). Par exemple :

A ET B = A · B = AB
A OU B = A + B
NON A = A'

13.1.1 Axiomes et théorèmes

Un axiome est une règle fondamentale que l'on doit tenir pour acquis. Le premier axiome stipule que le résultat de toute opération booléenne, appliquée à des variables booléennes, produit un résultat booléen (propriété de fermeture). Les opérateurs booléens sont définis par les tables de vérité de la figure 13.1.

X	Y	NOT X	X OU Y	X ET Y
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Figure 13.1 Fonctions booléennes

[1] Ainsi nommée en l'honneur du mathématicien anglais George Boole (1815-1864) qui en a proposé les principes de base dans son traité publié en 1854 : *An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities*

Les identités de base de l'algèbre de Boole comprennent la commutativité, l'associativité et la distributivité; elles sont données ci-dessous.

$1 \cdot A = A$	$0 + A = A$	$0 \cdot A = 0$
$1 + A = 1$	$A \cdot A' = 0$	$A + A' = 1$
$A \cdot A = A$	$A + A = A$	$A + B = B + A$
$A \cdot B = B \cdot A$	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	

Notez que la seconde loi de distributivité, qui est la dernière du lot ci-dessus, n'est pas valide en algèbre conventionnelle. À partir de ces axiomes, on peut déduire quelques théorèmes qui permettent de manipuler des expressions logiques de façon plus efficace que de revenir constamment aux axiomes. Prenons quelques exemples.

Théorème 1 : $X + XY = X$

Preuve

$X + XY = X1 + XY$ utilisant $1X = X$ et commutativité

$= X(1 + Y)$ utilisant distributivité

$= X(1)$ car $1+Y = 1$

$= X$

Théorème 2 : $X(X' + Y) = XY$

Preuve

$X(X' + Y) = XX' + XY$ multiplier par X

$= 0 + XY$ car $XX' = 0$

$= XY$

Théorème 3 : $X(X + Y) = X$

Preuve

$X(X + Y) = XX + XY$ multiplier par X

$= X + XY$ car $XX = X$

$= X$ par le théorème 1

Théorème 4 : $(X + Y)(X + Y') = X$

Preuve

$(X + Y)(X + Y') = XX + XY' + XY + Y'Y$ multiplier

$= X + XY' + XY$ car $XX = X$ et $YY' = 0$

$= X(1 + Y' + Y)$

$= X$

Théorème 5 : $XY + X'Z + YZ = XY + X'Z$

Preuve

$XY + X'Z + YZ = XY + X'Z + YZ(X + X')$ car $X + X' = 1$

$= XY + X'Z + XYZ + X'YZ$ multiplication

$= XY(1 + Z) + X'Z(1 + Y)$ distributivité

$= XY(1) + X'Z(1)$ $1 + Y = 1$

$= XY + X'Z$

Notez bien que toutes les variables dans un théorème peuvent être remplacées par d'autres variables ou expressions; ainsi, si X + XY = X, selon le théorème 1, X' + X'Y = X', si nous remplaçons X par X'.

De même, vous pouvez éviter certaines erreurs en faisant attention et en réfléchissant. Par exemple, X'Y' + XY n'est pas égal à 1, et ne peut pas être simplifié; il en est de même pour X'Y + XY'. Notez que (XY)' n'est pas égal à X'Y', et que(X+Y)' n'est pas égal à X' + Y'.

Exemple
La fonction OU exclusif donne un résultat vrai si les deux opérandes ont une valeur différente, et faux dans les autres cas. On veut montrer que cette fonction est associative, c'est-à-dire que :

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

Comme $X\oplus Y = XY'+X'Y$, on peut développer le terme de gauche de la façon suivante :

$A\oplus(B\oplus C) = A\oplus(BC'+B'C) = A(BC'+B'C)'+A'(BC'+B'C) = A(B'+C)(B+C')+A'BC'+A'B'C = AB'C'+ABC+A'BC'+A'B'C$

$(A\oplus B)\oplus C = (AB'+A'B)\oplus C = (AB'+A'B)C'+(AB'+A'B)'C = AB'C'+A'BC'+(A'+B)(A+B')C = AB'C'+A'BC'+(A'B'+AB)C$

$(A\oplus B)\oplus C = AB'C'+A'BC'+A'B'C+ABC$

On peut aussi montrer que toute fonction logique peut être réalisée par la fonction OU exclusif et la fonction ET. Prenons un exemple : $F = A\oplus B = AB'+A'B$. Si A=0 alors F = B car 0B'+1B = B. Si A = 1 alors F = B'. Si une entrée à une porte OU exclusif est connectée à un 1 logique, alors l'autre entrée apparaît à la sortie comme son complément. La porte OU exclusif agit donc comme inverseur; si on la connecte à la sortie d'une porte ET on obtient ce qu'on appelle une porte NAND; comme on peut tout engendrer avec des portes NAND (voir plus bas), il s'ensuit que l'on peut tout engendrer avec des portes OU exclusif et ET.

13.1.2 Lois de De Morgan

Deux théorèmes ont une importance particulière, il s'agit des lois de De Morgan[2] qui s'énoncent simplement :

$(ABC)' = A' + B' + C'$
$(A + B + C)' = A'B'C'$

Les axiomes et les divers théorèmes sont utilisés pour simplifier les expressions booléennes. Établissons un nouveau théorème en utilisant les lois de De Morgan.

Théorème 6 :	$(X + Y)(X' + Z)(Y + Z) = (X + Y)(X' + Z)$		
Preuve	$(X + Y)(X' + Z)(Y + Z)$	$= (((X + Y)(X' + Z)(Y + Z))')'$	complémenter 2 fois car X = (X')'
		$= ((X + Y)' + (X' + Z)' + (Y + Z)')'$	De Morgan sur produit intérieur
		$= (X'Y' + XZ' + Y'Z')'$	De Morgan sur chaque groupe
		$= (X'Y' + XZ')'$	par le théorème 5
		$= (X + Y)(X' + Z)$	De Morgan

[2] dues à Augustus de Morgan (1806-1871), mathématicien anglais.

13.1.3 Simplification d'expressions booléennes

Prenons quelques exemples d'expressions et du processus de simplification de ces expressions, l'objectif étant de réduire le nombre de termes, car, lorsque nous arriverons aux circuits on voudra minimiser le nombre de composants.

Exemple 1

A+B'+A'B+(A+B')A'B = A+B'+A'B+AA'B+A'B'B
= A+B'+A'B — X'X = 0
= A+B+B' — X+X'Y = X+Y
= 1 — X+X' = 1 et X+1=1

Exemple 2

(A+B')(A'+C)(B+C') = (AA'+AC+A'B'+B'C)(B+C')
= (AC+A'B'+B'C)(B+C') — X'X = 0
= (AC+A'B')(B+C') — par le théorème 5
= ABC+AC'C+A'B'B+A'B'C'
= ABC+A'B'C'

Exemple 3

ABD'+B'CD+ABC'+BCD+A'CD = ABD'+CD(B'+B+A')+ABC'
= ABD'+CD+ABC'
= AB(C'+D')+CD = AB(CD)'+CD et, par théorème 1 de 13.1.4 = AB+CD

Exemple 4

A'BD+AD+BCD'+A'BC = D(A'B+A)+BCD'+A'BC
= D(B+A)+ BCD'+A'BC — par théorème 1 de 13.1.4
= BD+AD+ BCD'+A'BC
= B(D+CD')+AD+A'BC
= B(C+D)+AD+A'BC — par théorème 1 de 13.1.4
= BC+BD+AD+A'BC
= BD+BC(1+A')+AD
= BD+BC+AD

Exemple 5

(A+B+C)(A'+B+C)(A'+B+C') = (B+C)(A'+B+C') — car (X+Y)(X+Y') = X
= C(A'+B)+BC' — car (X+Y)(X'+Z) = XZ+X'Y
= A'C+BC+BC'
= A'C+B(C+C')
= A'C+B

13.1.4 Exercices

1) Démontrez le théorème suivant : X+X'Y = X+Y.
2) Démontrez le théorème suivant : (X+Y)(X'+Z) = XZ+X'Y.
3) Démontrez le théorème suivant : (X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z).
4) Démontrez le théorème suivant : (XYZ)' = X'+Y'+Z'.

5) Simplifiez l'expression : X'YZ'+X'YZ+XY'Z+XYZ.
6) Simplifiez l'expression : ((X'Y)'(XZ)')'.
7) Simplifiez l'expression : (W+X+YZ)(W'+X)(X'+Y).

13.2 Portes logiques

Un ordinateur est composé d'un certain nombre de circuits logiques et l'algèbre de Boole est utilisée pour modéliser les circuits des dispositifs électroniques. Les éléments de base de tels circuits sont appelés *portes* et chaque type de porte réalise une opération booléenne. En utilisant ces portes, nous appliquerons les règles de l'algèbre de Boole, pour concevoir des circuits en mesure d'accomplir certaines tâches. Les circuits que nous étudierons d'abord donnent un signal de sortie qui dépend uniquement du signal d'entrée; de tels circuits n'ont pas de mémoire, on les appelle des *circuits combinatoires*.

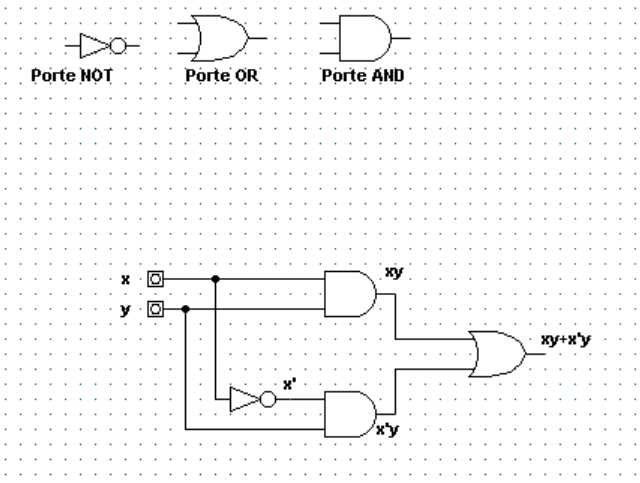


Figure 13.2 Portes NOT, AND, OR et circuit exemple

Les types de portes de base réalisent les fonctions de base de l'algèbre de Boole. On a ainsi des portes NON (NOT), ET (AND) et OU (OR), qui sont *fonctionnellement complètes*. Pour représenter chacun de ces types de porte, on utilise un symbole particulier, tel qu'illustré par la figure 13.2 (tracée au moyen du logiciel libre Digital Works[3] de David John Barker de l'université de Teesside en Angleterre).

Exemple

A	B	C	A+B'	B⊕C	B'C	ABC'	F	G
0	0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	1	1
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1	1
1	1	0	1	1	0	1	1	1
1	1	1	1	0	0	0	0	0

[3] http://www.spsu.edu/cs/faculty/bbrown/circuits/howto.html

Soit à tracer le diagramme des circuits générant F et G à partir des entrées a, b et c, selon les définitions suivantes : F = (A+B')(B⊕C) et G = B'C+ABC'. Le signe ⊕ représente la fonction OU exclusif, où le résultat est 1 si les deux opérandes sont différents. La table de vérité ci-dessus montre que F = G.
La figure 13.3 donne les diagrammes correspondants à ces deux fonctions.

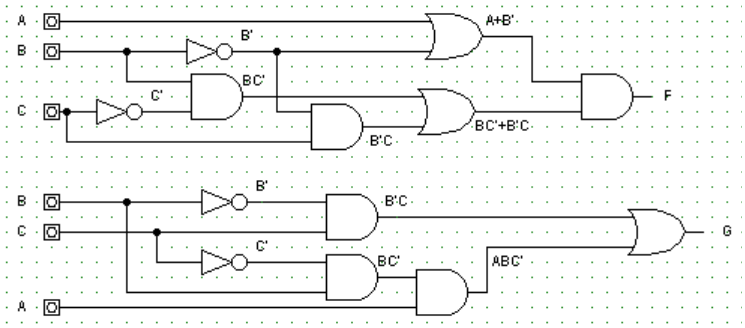


Figure 13.3 Circuits exemples pour les fonctions F et G

On peut considérer les portes ET et NON comme étant *fonctionnellement complètes*, à condition de pouvoir réaliser la fonction OU en les utilisant. On peut y arriver en appliquant la loi de De Morgan : X + Y = (X'Y')'. De la même façon, les opération OU et NON sont fonctionnellement complètes, car elles permettent de synthétiser l'opération ET : XY = (X'+Y')'.

Pour cette raison, il existe deux autres types de porte qui sont très utilisés : les portes NAND et NOR, dont les symboles sont illustrés par la figure 13.4. On notera que A NAND B = (A AND B)' et que A NOR B = (A OR B)'. Quelles sont les sorties des deux portes NAND du circuit de cette figure?

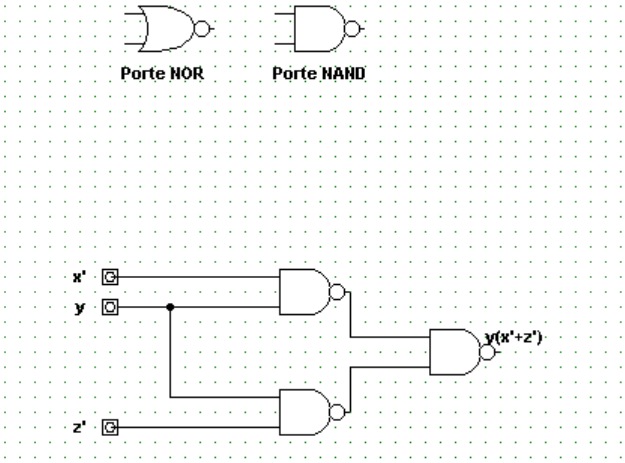


Figure 13.4 Portes NAND et NOR et circuit exemple

La porte NAND est plus rapide et moins coûteuse que la porte AND, et il en est de même pour la porte NOR et la porte OR. Dans ces conditions, il arrive que l'on veuille concevoir des circuits uniquement avec des portes NAND, ou uniquement avec des portes NOR. La figure 13.5 illustre la réalisation de la fonction X+Y+Z avec des portes NAND. Cette conversion utilise les lois de De Morgan, non plus pour simplifier l'expression, mais plutôt pour lui trouver un équivalent.

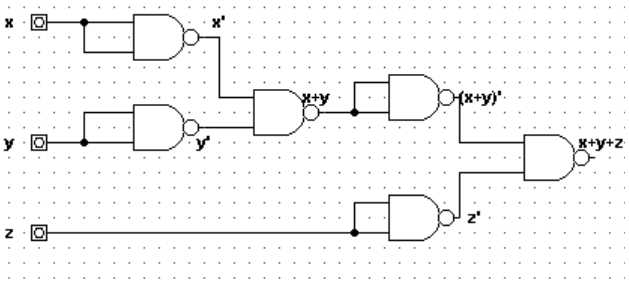


Figure 13.5 Fonction X+Y+Z réalisée en logique NAND

La figure 13.6 illustre la réalisation de la fonction XYZ avec des portes NOR.

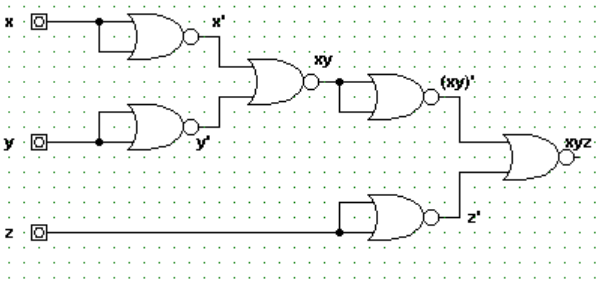


Figure 13.6 Fonction XYZ réalisée en logique NOR

13.3 Circuits combinatoires

Les circuits combinatoires sont des circuits dont la sortie ne dépend que de l'entrée, comme nous l'avons mentionné précédemment, et qui permettent de réaliser les fonctions essentielles de l'ordinateur.

13.3.1 Exemples de circuits

Multiplexeur

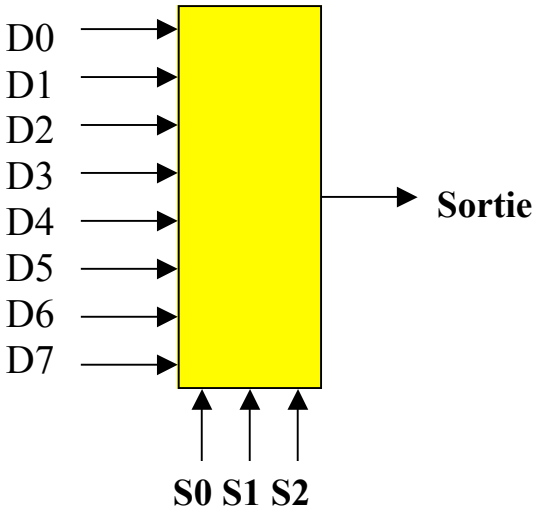


Figure 13.7 Multiplexeur 1 de 8

Un multiplexeur est un dispositif qui sélectionne une donnée d'entrée parmi toutes les données qu'il reçoit en parallèle. La figure 13.7 représente le diagramme de bloc d'un multiplexeur de huit entrées et de trois bits de sélection.

Les trois bits de sélection déterminent laquelle des entrées est passée à la sortie du multiplexeur. Ainsi, par exemple, si les entrées sont des bits représentant les 8 états possibles du processeur après certaines opérations, les bits de sélection peuvent servir à indiquer lequel des bits d'état est choisi et transmis par le multiplexeur. Ces bits de sélection pourraient venir de l'instruction en cours d'exécution, par exemple.

Démultiplexeur

Le démultiplexeur effectue la fonction inverse du multiplexeur, c'est-à-dire la conversion d'un code binaire de n bits en l'une des 2^n sorties possibles. La figure 13.8 représente le diagramme de bloc du démultiplexeur.

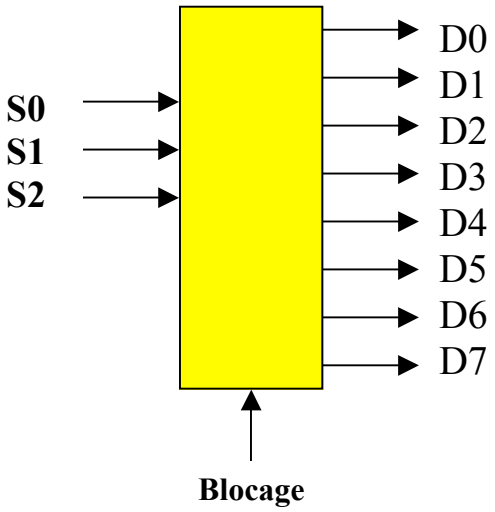


Figure 13.8 Démultiplexeur

Décodeur

Un décodeur est un dispositif qui accepte en entrée un code binaire et qui produit une sortie égale à 1 sur une des lignes de sortie et 0 sur les autres. Un décodeur est en fait un démultiplexeur! La figure 13.9 montre comment le circuit d'un démultiplexeur 3 lignes à 8 lignes peut être adapté en un décodeur d'instruction correspondant à la figure 13.8.

Pour chaque valeur de l'entrée, de 000 à 111, une des sorties w_n est mise à 1. Si le code de trois bits est le code binaire d'une instruction, alors les sorties peuvent être étiquetées par le symbole correspondant à l'instruction correspondante. La figure 13.10 donne une table de vérité pour un tel circuit, où une sortie valant 1 est associée à un code de trois bits particulier.

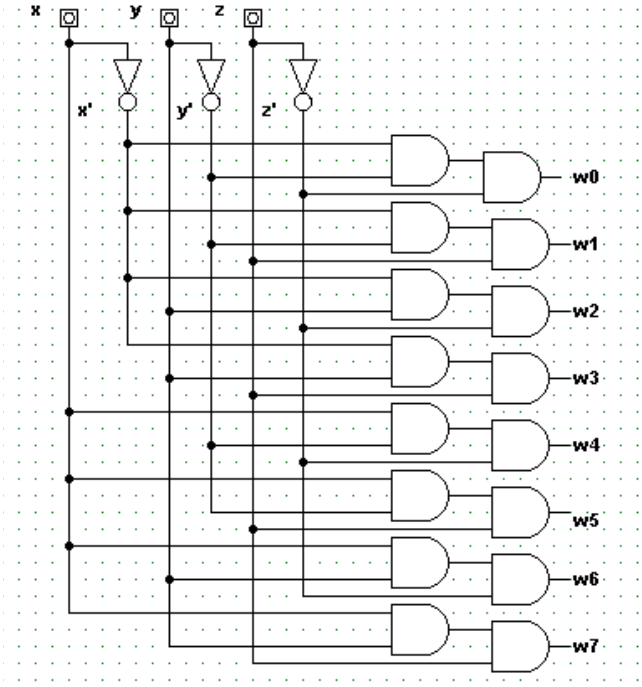


Figure 13.9 Décodeur d'instruction

X	Y	Z	W0	W1	W2	W3	W4	W5	W6	W7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Figure 13.10 Table de vérité du décodeur d'instruction

Par exemple, pour un processeur donné, les sorties W pourraient correspondre aux instructions suivantes : W0 = STOP, W1 = LDA, W2 = STA, W3 = ADDA, W4 = SUBA, W5 = BR, W6 = BEQ, W7 = BNE.

Additionneur

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 13.11 Table de vérité du demi-additionneur

Nous voulons maintenant montrer comment les circuits logiques peuvent être utilisés pour effectuer des opérations arithmétiques sur des entiers à partir de leur représentation binaire. Nous commençons par un circuit permettant d'additionner x et y, où x et y sont des bits. La sortie du circuit sera composée de 2 bits, un pour la somme et l'autre pour la retenue. On appelle un tel circuit un demi-additionneur. La figure 13.11 donne la table de vérité associée au demi-additionneur.

On déduit de cette table que c = xy, car la valeur 1 de c apparaît pour les valeurs 1 de x et de y; puis s = x'y + xy', car les valeurs 1 de s apparaissent pour x=0 et y=1 (x'y) ou pour x=1 et y=0 (xy'), ce qui donne s = (x+y)(x'+y') ou (x+y)(xy)'. La figure 13.12 illustre le circuit du demi-additionneur correspondant.

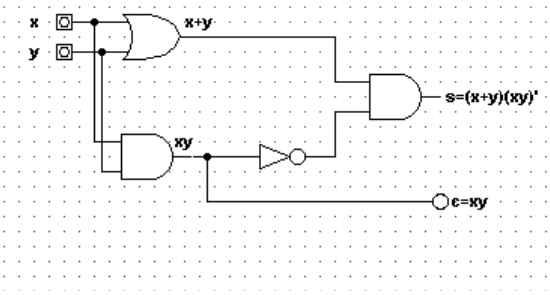


Figure 13.12 Demi-additionneur

On utilise un additionneur complet lorsqu'on additionne deux bits et une retenue pour obtenir, comme dans le cas du demi-additionneur, deux bits en sortie, la somme et la retenue. La table de vérité d'un additionneur complet se trouve à la figure 13.13.

x	y	ci	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 13.13 Table de vérité d'un additionneur complet

La figure 13.14 montre comment composer un additionneur complet à partir de demi-additionneurs (ne tenez pas compte des lettres dans les blocs des demi-additionneurs).

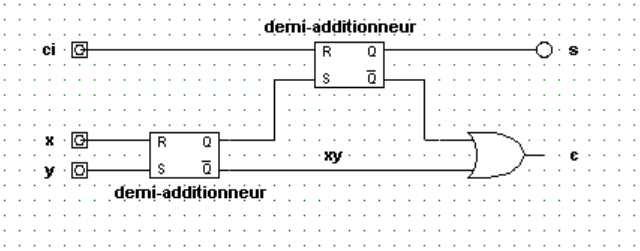


Figure 13.14 Un additionneur complet

La figure 13.15 montre comment nous pouvons combiner des demi-additionneurs et des additionneurs complets, pour effectuer l'addition de deux entiers de trois bits.

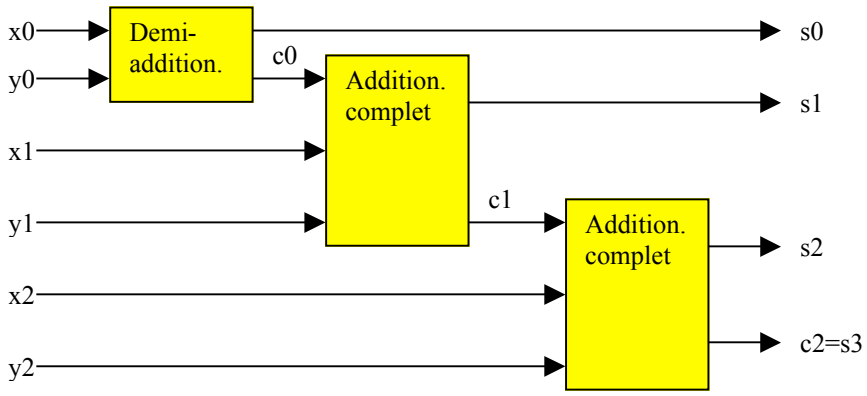


Figure 13.15 Addition de deux nombres de trois bits

Multiplicateur

Pour donner un exemple un peu plus compliqué, sans qu'il ne le soit trop, montrons comment nous pouvons définir un multiplicateur de deux quantités de deux bits chacune. La table de vérité définissant ce multiplicateur se trouve à la figure 13.16; A_0 et A_1 sont les deux bits du multiplicande et B_0 et B_1 sont les deux bits du multiplicateur.

A1	A0	B1	B0	W3	W2	W1	W0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Figure 13.16 Table de vérité du multiplicateur

Le multiplicateur accepte donc quatre données d'entrée et produit quatre données de sortie; nous ne montrerons pas le diagramme de bloc correspondant, qui est fort simple. En reportant pour chaque colonne de la sortie les termes qui correspondent à une sortie égale à 1, nous obtenons les équations booléennes définissant les sorties (où les variables sont An, Bn et Wn).

W0 = A1'A0B1'B0+A1'A0B1B0+A1A0B1'B0+A1A0B1B0

W1 = A1'A0B1B0'+A1'A0B1B0+A1A0'B1'B0+A1A0'B1B0+A1A0B1'B0+A1A0B1B0'
W2 = A1A0'B1B0'+A1A0'B1B0+A1A0B1B0'
W3 = A1A0B1B0

Il nous faut maintenant réduire ces équations ; nous pouvons utiliser les axiomes et théorèmes de l'algèbre de Boole ou une autre méthode de simplification (voir section suivante). Voici les équations réduites obtenues par simplification.

W0 = A0B0
W1 = A1'A0B1+A0B1B0'+A1A0'B0+A1B1'B0
W2 = A1A0'B1+A1B1B0'
W3 = A1A0B1B0

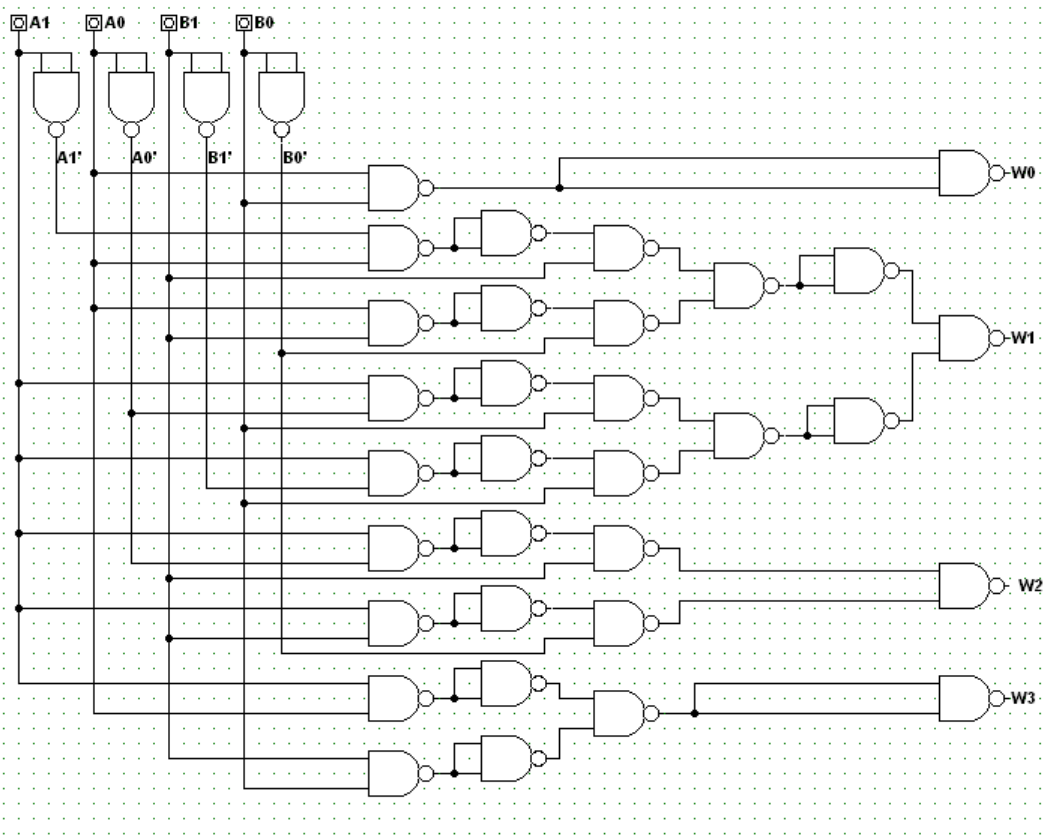


Figure 13.17 Multiplicateur réalisé par logique NAND

À partir de ces équations simplifiées nous pouvons tracer le circuit correspondant. Cependant, les portes NAND opèrent à plus grande vitesse que les portes AND et peuvent aussi être fabriquées avec un nombre de composants réduit (au niveau de la puce). Il est donc intéressant de considérer réaliser le circuit équivalent au moyen de portes NAND. Pour ce faire, nous devons modifier les équations obtenues en appliquant les lois de De Morgan, non plus pour simplifier, mais pour obtenir des produits et des compléments. Nous prenons le complément du complément de chaque expression, ce qui est, bien entendu, égal à l'expression de départ, et nous conservons les produits et les compléments. Ainsi :

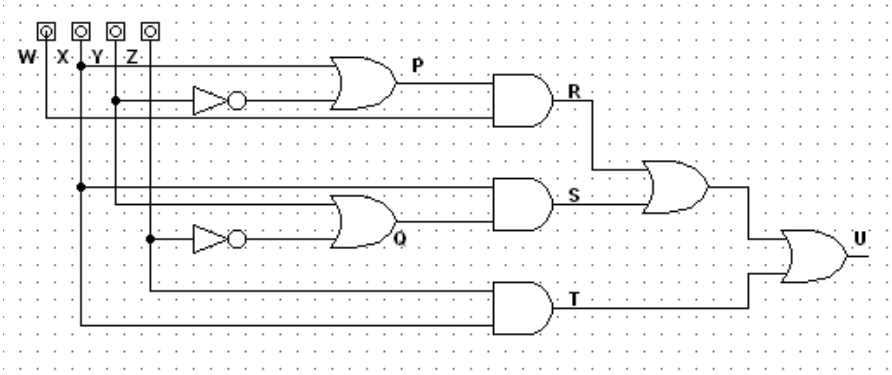
W0 = A0B0 = ((A0B0)')'

W1 = ((A1'A0B1)'(A0B1B0')'(A1A0'B0)'(A1B1'B0)')'
W2 = ((A1A0'B1)'(A1B1B0')')'
W3 = ((A1A0B1B0)')'

La figure 13.17 montre la réalisation de ce circuit avec la logique NAND.

13.3.2 Exercices

1) Étant donné le diagramme ci-dessous, établissez une table de vérité pour les valeurs des quatre variables et les valeurs de P, Q, R, S, T et U.



2) Tracez le circuit correspondant à l'expression F= (ST+UV+WX+YZ)'.
3) Tracez le circuit correspondant à l'expression F= X'Y+XZ' en logique NAND.
4) Au moyen des axiomes et théorèmes de l'algèbre de Boole, effectuez complètement la simplification des expressions de W0, W1, W2 et W3 du multiplicateur.

13.3.3 Simplification des circuits par tables de Karnaugh[4]

Lorsqu'on doit simplifier une expression booléenne de plusieurs variables, il est parfois un peu difficile de procéder directement par l'algèbre de Boole. La table de Karnaugh est un moyen commode de représenter une fonction booléenne de trois, quatre, ou même au maximum, 6 variables. La table est un tableau de 2^n carrés, qui représentent les combinaisons possibles des valeurs de n variables. Pour deux variables, la table de Karnaugh aura 4 éléments; pour 3 variables, la table de Karnaugh aura 8 éléments, pour quatre variables, la table de Karnaugh aura 16 éléments et pour 5 variables elle en aura 32, et 64 pour 6 variables. La taille de la table de Karnaugh est ce qui limite cette méthode graphique, car une table de 64 éléments est difficile à bien appréhender. Les combinaisons des variables repérant les éléments de la table diffèrent d'un seul bit avec leurs voisines, ce qui oblige à faire attention à la façon dont on remplit la table. La figure 13.18 illustre des tables de différentes tailles. Les valeurs 1 dans la table indiquent la présence du terme correspondant. Ainsi, la table de 2 variables représente la fonction a'b+ab; la table de trois variables représente la fonction a'b'c+ab'c+abc; la table de quatre variables représente la fonction a'bc'd+a'bcd+abc'd+ab'c'd'+ab'cd'.

[4] Maurice Karnaugh (1924-), mathématicien américain ayant travaillé aux Bell Laboratories et pour la compagnie IBM.

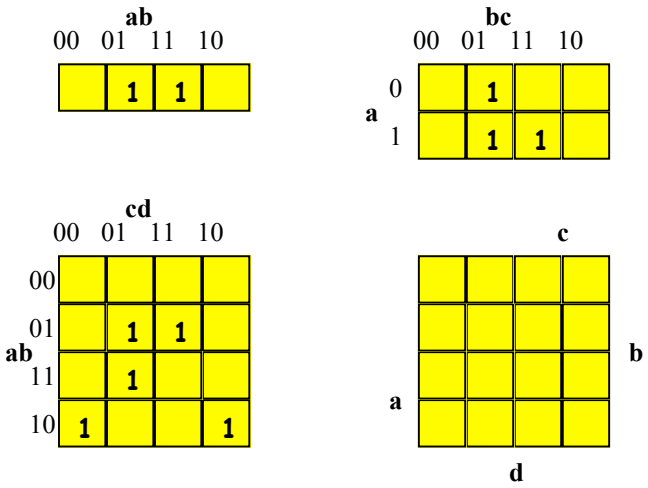


Figure 13.18 Tables de Karnaugh

La table vide de la figure 13.18 représente une autre façon de grouper les quatre variables : la variable a correspond aux deux rangées du bas, les deux rangées du haut correspondant à son complément. La variable b correspond aux deux rangées du milieu, les autres rangées correspondant à son complément. La variable c correspond aux deux colonnes de droite et c' aux deux colonnes de gauche, tandis que la variable d correspond aux deux colonnes du milieu et aux deux colonnes externes. Avec un peu de pratique, l'interprétation de ces tables deviendra plus claire. La figure 13.19 illustre en turquoise les rangées ou colonnes correspondant à l'une des quatre variables, les cases jaunes correspondant au complément de la variable.

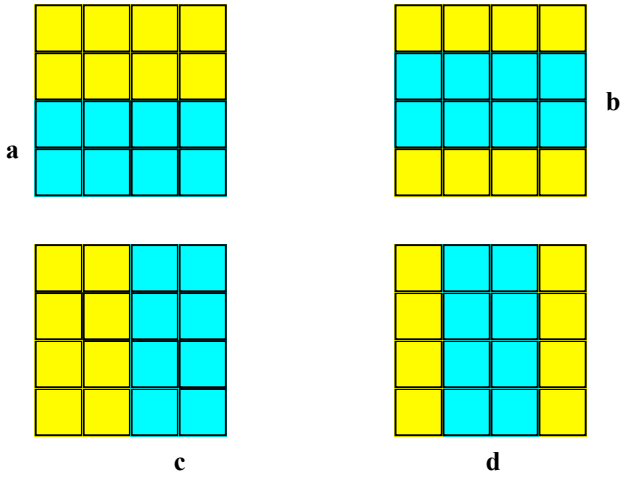


Figure 13.19 Zones des tables de Karnaugh pour 4 variables

Lorsque la fonction à simplifier est exprimée en une somme de termes, comprenant chacun toutes les variables, on reporte tous les termes dans une table de Karnaugh, en plaçant des 1 dans les cases correspondant aux éléments. Ensuite, on essaye de regrouper les valeurs 1 voisines. En effet, si deux cases adjacentes contiennent une valeur 1, les deux termes en cause ne diffèrent que d'une variable. On peut alors les grouper en éliminant cette variable. Ainsi, dans la table de deux variables de la figure 13.18, on groupe les deux termes en éliminant les variables qui différent (a et a') et on conserve b, qui

est la simplification de l'expression donnée plus haut. De même, toujours dans la même figure, la table de trois variables permet deux regroupements : a'b'c et ab'c qui se simplifient en b'c, puis ab'c et abc qui se simplifient en ac, regroupement qui recoupe le premier, donnant l'expression simplifiée : ac+b'c. La table de quatre variables de cette même figure permet trois regroupements; les deux premiers se recoupent et se trouvent dans le milieu de la table, le troisième est moins évident. En effet, les cases extrêmes de la table sont voisines : la première rangée est voisine de la dernière rangée et, de la même façon, la première colonne est voisine de la dernière colonne. Donc les regroupements sont les suivants : a'bc'd et a'bcd qui se réduisent à a'bd; a'bc'd et abc'd qui se réduisent à bc'd; ab'c'd', et ab'cd' qui se réduisent à ab'd'. L'expression simplifiée est donc a'bd+bc'd+ab'd'.

Lorsqu'on cherche les regroupements, il nous faut trouver **les plus grands regroupements possibles**, la taille des blocs regroupés étant 1, 2, 4, 8, 16. On cherche aussi à **minimiser le nombre de ces blocs**. Un élément de la table peut appartenir à plus d'un regroupement.

Exemple 1
Soit à simplifier la fonction suivante : f = abc+abc'+ab'c+ab'c'+a'bc+a'b'c+a'b'c'
La table de Karnaugh correspondant à cette équation est donnée par la figure 13.20.

a \ bc	00	01	11	10
0	1	1	1	
1	1	1	1	1

Figure 13.20 Table de Karnaugh à trois variables

Nous choisissons les 3 groupements suivants : la rangée du bas (a), le carré de quatre éléments à gauche (b') et le carré du centre (c). Ceci nous donne l'expression simplifiée a+b'+c.

Exemple 2
Prenons maintenant une expression avec quatre variables et simplifions là par la méthode de Karnaugh. La fonction à simplifier étant : f = abc'd'+ab'cd+ab'cd'+ab'c'd'+a'bc'd'+a'b'cd'+a'b'c'd', la table de Karnaugh est donnée par la figure 13.21.

a \ cd	00	01	11	10
00	1			1
01	1			
11	1			
10	1		1	1

b

Figure 13.21 Table de Karnaugh à 4 variables

Nous choisissons les regroupements suivants : la colonne de gauche (c'd'), les deux cases du bas à droite (ab'c) et les deux coins de droite (b'cd'). Ceci conduit à l'expression simplifiée c'd'+ab'c+b'cd'. Notez cependant que ce n'est ni la seule possibilité, ni la meilleure : on aurait pu regrouper les quatre coins dans le terme b'd', qui donne l'expression un peu plus simplifiée c'd'+ab'c+b'd'.

Exemple 3
Utilisons une table de Karnaugh pour convertir une somme de produits en un produit de sommes. Soit à convertir la fonction f = abc+c'd+a'bd; nous plaçons les termes dans une table de Karnaugh, illustrée par la partie gauche de la figure 13.22. Nous savons que les lois de De Morgan nous permette de prendre le complément d'une expression et de transformer une somme de produits en un produit de sommes.

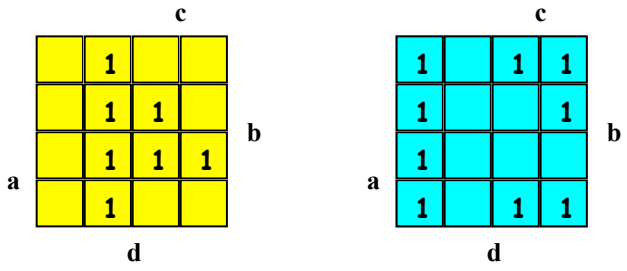


Figure 13.22 Tables de Karnaugh pour conversion

Le complément de la fonction est donné par la table de Karnaugh de droite dans la figure 13.22. L'expression correspondante est alors constituée en choisissant les blocs de la colonne de gauche (c'd'), du carré constitué des deux cases du haut à droite et des deux cases du bas à droite (b'c) et du carré constitué des deux éléments du haut de la colonne de droite et de la colonne de gauche (a'd'). L'expression du complément est donc : f' = c'd'+b'c+a'd'. Nous retrouvons f en prenant le complément de f', soit : f = (c'd'+b'c+a'd')', ce qui nous donne : (c+d)(b+c')(a+d) qui est le résultat cherché.

Exemple 4
Utilisez les tables de Karnaugh pour simplifier les expressions du multiplicateur de la page 195.
Conditions neutres

Ch	Fr	Hum	Sec	Cf	Cl	Dh	Hm
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	X	X	X	X
0	1	0	0	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	x

Figure 13.23 Table de vérité pour le contrôleur de température et d'humidité

Il existe des problèmes pour lesquels la table de vérité n'est pas totalement spécifiée. Ceci se produit de temps en temps pour des circuits dans lesquels certaines combinaisons des valeurs d'entrée ne peuvent se produire. Dans ces cas, la sortie peut être soit vraie, soit fausse, sa valeur ne nous intéresse pas, elle est considérée neutre.

Pour illustrer ceci, nous prendrons l'exemple d'un contrôleur de température et d'humidité pour un logement. Un palpeur indiquera si la température est inférieure à 14 degrés Celsius; un second palpeur indiquera si la température est supérieure à 25 degrés; un troisième palpeur indiquera si l'humidité est supérieure à 80% et un dernier palpeur indiquera si l'humidité est inférieure à 30%. Il est clair que certaines combinaisons d'entrée ne peuvent se produire; les palpeurs de température ne peuvent tous deux être à 1, pas plus que les palpeurs d'humidité. Le système fournira quatre sorties qui contrôlent chacune un appareil : chauffage, climatisation, déshumidificateur et humidificateur. La figure 13.23 présente la table de vérité de ce système.

On voit quelles sont les conditions de déclenchement du chauffage, du climatiseur, du déshumidificateur et de l'humidificateur; on remarque qu'il existe un certain nombre d'états qui sont interdits, comme par exemple démarrer le chauffage et le climatiseur. La figure 13.24 présente les tables de Karnaugh correspondant à cette situation (une table par sortie).

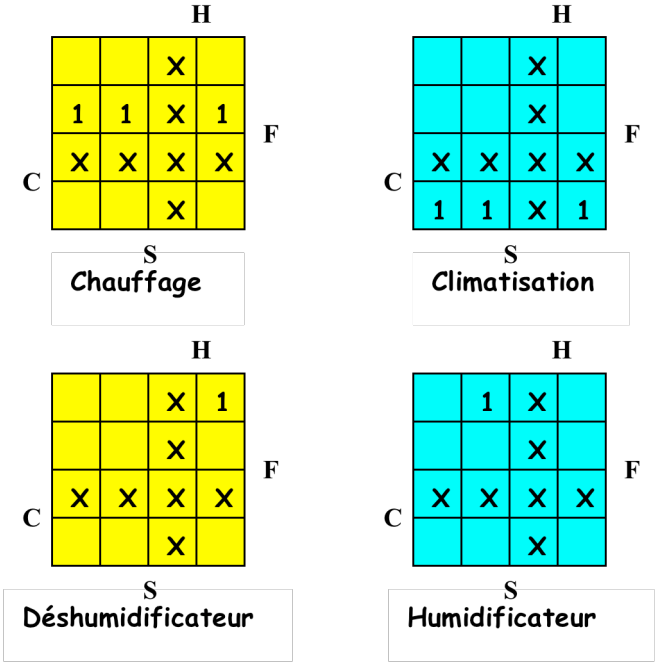


Figure 13.24 Tables de Karnaugh pour le contrôleur

En groupant les termes des différentes tables en incluant les conditions neutres pour obtenir les plus grands groupes, on obtient les expressions réduites suivantes.

Chauffage = F
Climatisation = C
Déshumidificateur = C'F'H
Humidificateur = F'C'S

13.3.3.1 Exercices

1) Montrez la minimisation du circuit a'b'c'd'+a'b'c'd+a'b'cd'+a'bc'd+ab'c'd'+ab'c'd+ab'cd'+abc'd au moyen de la table de Karnaugh.
2) Montrez la minimisation du circuit suivant par table de Karnaugh a'b'c'd'+a'b'c'd+a'b'cd'+a'b'cd+a'bc'd+a'bcd'+a'bcd+ab'c'd'+ab'c'd+abc'd'+abc'd+abcd'.
3) Un circuit logique possède quatre entrées A, B, C et D; ces données représentent deux paires de bits (A,B) et (C,D). On soustrait les bits (C,D) des bits (A,B) pour donner un résultat (Q,R) et un signe S pour le résultat (1 si négatif). Établissez la table de vérité pour ce circuit et simplifiez les expressions des trois sorties par tables de Karnaugh.

13.3.4 Simplification par la méthode de Quine-McCluskey[5]

Lorsqu'il y a plus de quatre variables, la méthode de Karnaugh devient malcommode; de plus, la méthode s'appuie sur une inspection visuelle pour l'identification des termes groupés. Pour ces raisons, on a besoin d'une procédure pour simplifier les sommes de produits, qui puisse être mécanisée. La méthode de Quine-McCluskey (http://en.wikipedia.org/wiki/Quine-McCluskey_algorithm) répond à ce besoin et peut être programmée (http://cheeseshop.python.org/pypi/qm/0.1). Cette méthode a été développée dans les années cinquante du siècle dernier et comporte deux parties : la première partie détermine quels termes sont candidats à être inclus dans une somme de produits minimale; la seconde partie détermine quels termes finalement retenir.

Les étapes de la méthode sont les suivantes.

1. Transformer chaque terme en n variables de l'expression en une valeur binaire, 0 représentant un complément et 1 la variable.
2. Grouper les chaînes de bits selon le nombre de bits 1 qu'elles contiennent.
3. Déterminer tous les produits en n-1 variables que l'on peut former en effectuant la somme booléenne des termes de l'expansion. Les termes que l'on peut combiner ont des chaînes de bits qui diffèrent exactement en une position. Les bits différents sont remplacés par un tiret dans le nouveau terme.
4. Déterminer tous les produits de n-2 variables que l'on peut former en effectuant la somme booléenne des produits de n-1 variables obtenus à l'étape précédente. Les produits que l'on peut combiner possèdent un tiret dans la même position et diffèrent exactement en une position.
5. Répéter ce processus tant que c'est possible.
6. Trouver tous les produits booléens formés qui n'ont pas été utilisés pour former un nouveau produit booléen.
7. Trouver le plus petit ensemble de ces produits booléens, de sorte que la somme de ces produits représente la fonction booléenne. On y arrive en formant une table qui montre quels termes sont couverts par quels produits; tous les termes doivent être couverts par un produit. Cette étape est la plus difficile, on peut la programmer par une méthode avec retour-arrière.

Exemple
Soit à simplifier la fonction f = abcd'+ab'cd+ab'cd'+a'bcd+a'bc'd+a'b'cd+a'b'c'd.

[5] Edward J. McCluskey (1929-), ingénieur américain ayant réalisé cette méthode alors qu'il était étudiant de doctorat au M.I.T. et ayant travaillé aux Bell Laboratories avant d'être professeur à Princeton et plus tard à Stanford. Willard Van Orman Quine (1908-2000), logicien américain, professeur de philosophie à Harvard pendant 64 ans.

Les termes sont représentés par des chaînes de bits, puis classés par nombre de bits 1, c'est-à-dire 1 bit, 2 bits et 3 bits. La figure 13.25 illustre la méthode.

Termes et chaînes		Étape 3		Étape 4	
1 a'b'c'd	0001	(1,4) a'b'd	00-1	(1,3,4,7) a'd	0--1
2 ab'cd'	1010	(1,3) a'c'd	0-01		
3 a'bc'd	0101	(4,7) a'cd	0-11		
4 a'b'cd	0011	(3,7) a'bd	01-1		
5 abcd'	1110	(4,6) b'cd	-011		
6 ab'cd	1011	(2,6) ab'c	101-		
7 a'bcd	0111	(2,5) acd'	1-10		

Figure 13.25 Méthode de Quine-McCluskey

À l'étape 3, nous avons combiné tous les produits de la première colonne de la table. À l'étape 4, nous n'avons pu combiner que les 4 premiers termes de la colonne (en turquoise); il reste donc trois termes de cette colonne non combinés. Le terme de la troisième colonne est seul et ne peut être combiné. Par conséquent, nous terminons l'étape 5 avec quatre termes non utilisés, soit a'd, acd', ab'c et b'cd. La table de la figure 13.26 montre quels termes sont couverts par les termes réduits retenus.

	abcd'	ab'cd	a'bcd	ab'cd'	a'bc'd	a'b'cd	a'b'c'd
a'd			X		X	X	X
acd'	X			X			
ab'c		X		X			
b'cd		X				X	

Figure 13.26 Couverture des termes

Nous devons garder les termes a'd et acd', puisqu'ils sont les seuls à couvrir a'bcd, a'bc'd et a'b'c'd, pour le premier, et abcd', pour le second. Nous pouvons ensuite retenir un seul des deux termes restants pour le terme ab'cd. La solution est donc, soit a'd+acd'+ab'c, soit a'd+acd'+b'cd.

13.3.5 Exercices

1) Appliquez la méthode de Quine-McCluskey à l'expression : abc+ab'c+a'bc+a'b'c+a'b'c'
2) Appliquez la méthode de Quine-McCluskey à l'expression : abcd+abc'd+abc'd'+ab'cd+a'bcd+a'bc'd+a'bcd'+a'b'c'd

13.4 Circuits séquentiels

Les circuits combinatoires sont des circuits dont la sortie ne dépend que de l'entrée. Cependant, à part le cas de la mémoire morte (ROM), ils ne fournissent aucune mémoire ou information d'état, éléments qui sont aussi essentiels au fonctionnement d'un ordinateur. À ces fins, on utilise une forme plus complexe de circuits logiques, les circuits séquentiels. La sortie courante d'un circuit séquentiel dépend non seulement de l'entrée courante, mais aussi de l'historique des entrées passées. On dit généralement que la sortie courante d'un circuit séquentiel dépend de l'entrée courante et de l'état courant de ce circuit.

La forme la plus simple de circuit séquentiel est la bascule, un dispositif bistable qu'on peut apparenter à un bit de mémoire, et qui possède deux sorties qui sont toujours le complément l'une de l'autre.

13.4.1 La bascule S-R

La figure 13.27 illustre un circuit commun, appelé bascule S-R, qui possède deux entrées, S (*set*) et R (*reset*), et deux sorties habituellement notées Q et Q'. Le circuit est réalisé au moyen de deux portes NOR avec feed-back. Si on suppose que S, R et Q valent 0, les entrées de la porte inférieure sont toutes deux nulles, ce qui donne une sortie égale à 1; les entrées de la porte supérieure sont alors 1 et 0, ce qui produit une sortie nulle. L'état du circuit restera stable tant que S et R resteront nuls.

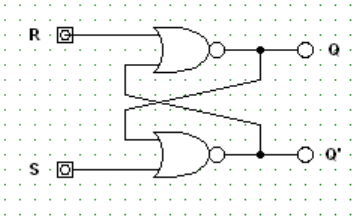


Figure 13.27 Bascule S-R

Ce circuit fonctionne comme une mémoire de 1 bit; la sortie Q est la valeur du bit, les entrées servant à « écrire » la valeur en mémoire. Dans l'état stable S=0, R=0, Q=0, Q'=1, si S change de valeur et passe à 1, les entrées de la porte inférieure sont 1 et 0 et, après un délai Δt, la sortie de la porte NOR sera Q'=0. Les entrées de la porte supérieure sont alors toutes les deux nulles; après un autre délai Δt, la sortie devient Q=1. Les entrées de la porte inférieure sont maintenant toutes les deux 1, ce qui laisse la sortie Q'=0 et la bascule est à nouveau dans un état stable. Même si S redevient 0, la sortie ne change pas.

L'entrée R effectue la fonction inverse. Lorsque R devient 1, il force Q à 0 et Q' à 1, quel que soit l'état précédent. À nouveau, il faut un délai de $2\Delta t$ avant que la stabilité soit retrouvée. Les circuits séquentiels n'ont pas une table de vérité qui leur est associée, comme les circuits combinatoires; à la place, ils ont une table caractéristique qui spécifie l'état du dispositif après une pulsion d'horloge, comme l'illustre la figure 13.28 pour la bascule S-R.

S(t)	R(t)	Q(t)	Q(t+1)	Condition
0	0	0	0	Stable
0	0	1	1	Stable
0	1	0	0	Reset
0	1	1	0	Reset
1	0	0	1	Set
1	0	1	1	Set
1	1	0	-	Non défini
1	1	1	-	Non défini

Figure 13.28 Table caractéristique de la bascule S-R

Comme le temps joue un rôle dans le comportement d'une bascule, on utilise souvent un diagramme temporel, permettant d'illustrer les changements dans le temps des entrées et des sorties de la bascule. La figure 13.29 est un diagramme temporel pour la bascule S-R.

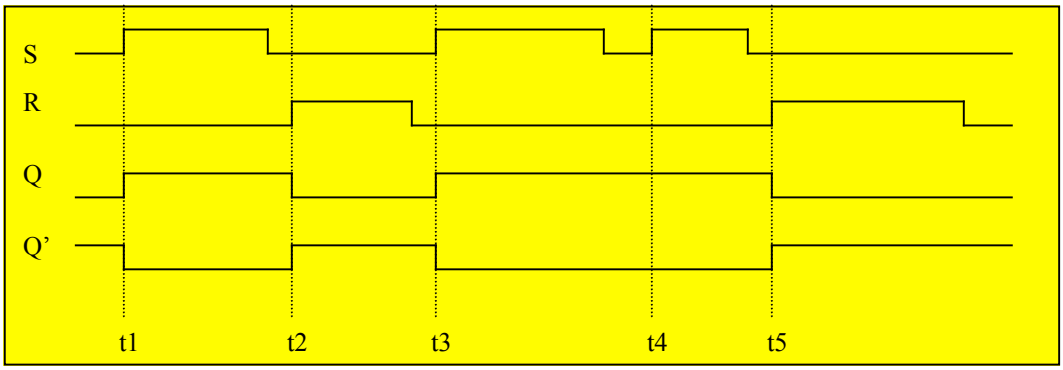


Figure 13.29 Diagramme temporel pour la bascule S-R

Au début, R, S et Q valent zéro; au temps t1, S devient 1 et, en conséquence, Q devient 1 (après un délai de porte trop petit pour l'échelle du diagramme temporel); S revient à zéro sans effet sur le système. Au temps t2, R devient 1, ce qui a pour effet de remettre Q à zéro; lorsque R redevient nul le système n'est pas affecté. Au temps t3, S reprend la valeur 1, ce qui provoque un changement de Q; lorsque S revient à zéro, puis, au temps t4, revient à 1, le système n'est pas affecté. Au temps t5, R reprend la valeur 1, qui remet Q à zéro.

L'opération de la bascule S-R est asynchrone : la sortie ne change que lorsque l'entrée change. Dans un ordinateur, il existe de très nombreux dispositifs qui changent d'état constamment. Afin de les contrôler pour permettre un fonctionnement ordonné, l'ordinateur comprend une horloge et les dispositifs changent d'état tous en même temps : ils sont synchronisés aux pulsions de l'horloge. La figure 13.30 montre une bascule S-R synchronisée; les entrées R et S ne pénètrent le dispositif que pendant une pulsion d'horloge.

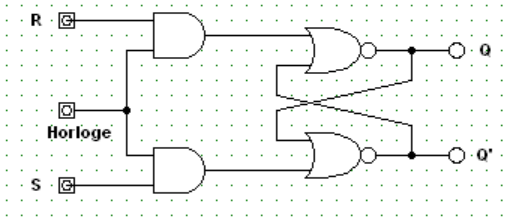


Figure 13.30 Bascule S-R synchronisée

13.4.2 Autres bascules (flip-flops)

Bascule D

Pour la bascule S-R, on doit éviter la combinaison R=1 et S=1; une façon de faire cela est de réduire les entrées à une seule comme le montre la figure 13.31 qui représente une bascule D (*data* ou *delay*). Elle est ainsi appelée, car elle fournit la mémoire pour un bit de données. La sortie de la bascule D est toujours égale à la plus récente valeur envoyée en entrée. On parle de délai, car la bascule retarde l'application de l'entrée à la prochaine pulsion d'horloge.

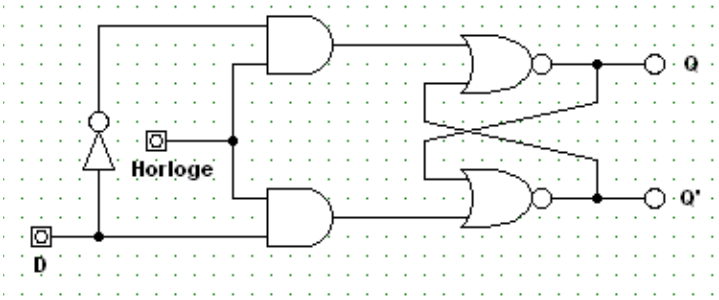


Figure 13.31 Bascule D

Bascule J-K

Une autre bascule utile est la bascule J-K qui, comme la bascule S-R, possède deux entrées. Dans le cas de cette bascule, toutes les combinaisons d'entrée sont valides. La figure 13.32 donne la table caractéristique de cette bascule.

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

Figure 13.32 Table caractéristique de la bascule J-K

La bascule J-K complète la bascule S-R en permettant la condition d'entrée 11 pour laquelle on passe d'un état à l'autre; si l'état est 0 il passe à 1 et inversement. On peut réaliser une bascule J-K au moyen d'une bascule S-R en ajoutant un circuit combinatoire pour alimenter les entrées S et R. Pour ce faire, on construit la table caractéristique de la figure 13.33 où on place dans les trois premières colonnes toutes les combinaisons de J, K et Q(t), avec les valeurs correspondantes de Q(t+1), S et R. Dans cette table les valeurs de Q(t+1) sont reprises de la table caractéristique de la figure 13.32. Les valeurs de S et de R sont déduites de la table caractéristique de la bascule S-R.

J	K	Q(t)	Q(t+1)	S	R
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	x	0
1	1	0	1	1	0
1	1	1	0	0	1

Figure 13.33 Table caractéristique de la bascule J-K basée sur une bascule S-R

Si on construit une table de Karnaugh à trois variables (Q, J, K) pour chaque entrée S et R en incluant les conditions neutres, on arrive aux expressions réduites suivantes : S = JQ' et R = KQ, ce qui conduit à la figure 13.34, qui illustre le circuit de la bascule J-K utilisant une bascule S-R.

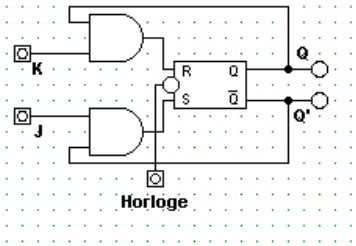


Figure 13.34 Circuit de la bascule J-K utilisant une bascule S-R

13.4.3 Registres

Pour illustrer l'utilisation des bascules dans l'ordinateur, nous prendrons l'exemple des registres. Un registre est un circuit numérique de l'unité centrale de traitement qui emmagasine un certain nombre de bits (16 pour PEP 8, 32 pour la plupart des processeurs). Il y a deux sortes de registres : les registres parallèles et les registres de décalage.

Les registres parallèles comprennent un ensemble de mémoires de 1 bit que l'on peut accéder en lecture ou en écriture simultanément. On y conserve des données. On peut les réaliser avec différentes bascules. La figure 13.35 montre une réalisation d'un registre de 4 bits avec des bascules S-R, dont l'avantage est de permettre une remise à zéro instantanée. On pourrait aussi utiliser des bascules D, sans l'avantage de la remise à zéro.

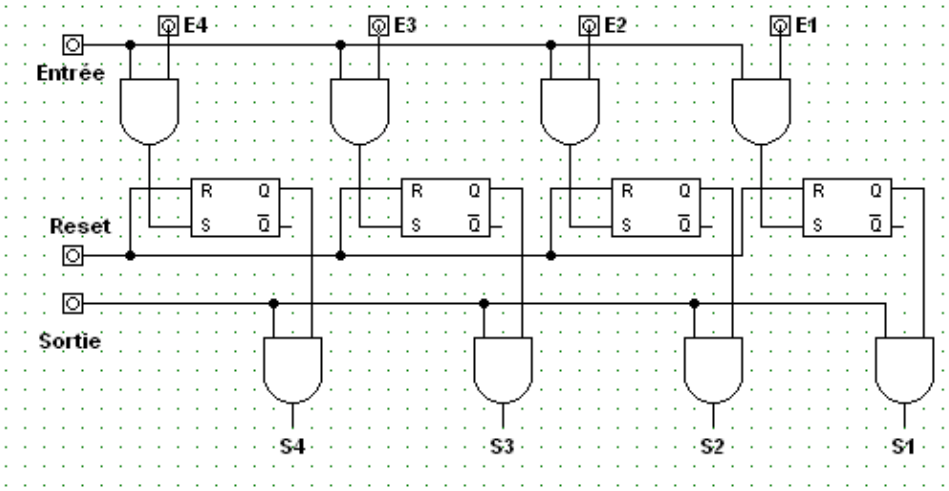


Figure 13.35 Registre parallèle de 4 bits

Les registres de décalage acceptent ou transfèrent de l'information de façon sérielle. La figure 13.36 illustre un tel registre de décalage de 4 bits réalisé au moyen de bascules D. Les données entrent à gauche et, à chaque pulsion d'horloge, sont décalées d'une position vers la droite, le dernier bit à droite étant transféré à l'extérieur.

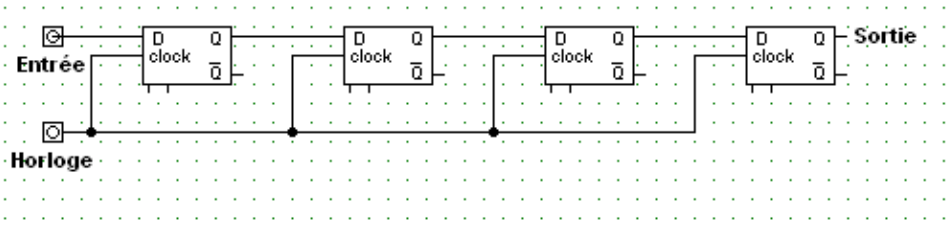


Figure 13.36 Registre de décalage de 4 bits

On peut utiliser les registres de décalage comme interface vers les dispositifs d'entrée-sortie sériels et également pour effectuer les décalages logiques, arithmétiques ou cycliques des registres. On utilise aussi un registre de décalage pour convertir un mot parallèle de n bits en un mot série de n bits consécutifs.

13.4.4 Exercices

1. Donnez un exemple de diagramme de temps pour la bascule D.
2. Donnez le circuit complet d'une bascule J-K.

