CI 8 – SED – Systèmes à événements discrets

Commande et comportement des systèmes combinatoires et séquentiels

Chapitre 1 – Étude des systèmes combinatoires

D'après ressources de David Violeau et Florestan Mathurin

Un des objectifs de l'automaticien est de concevoir la partie commande qui traite les informations et élabore les ordres. Ce cours étudie le cas où les informations et les ordres élaborés sont des variables binaires.

Problématique

Problématique :

Savoir

Savoirs :
- Exprimer le fonctionnement d'un système combinatoire sous forme d'équations logiques
- Optimiser la représentation logique du système
- Décrire, au moyen d'une représentation adaptée, le comportement du système

1 Numération et codage . . . 2
1.1 Système de numération . . . 2
1.2 Codage de l'information . . . 3
2 Systèmes combinatoires . . . 6
2.1 Définitions . . . 7
2.2 Algèbre de Boole . . . 9
3 Fonctions logiques . . . 13
3.1 Table de vérité . . . 14
4 Réorganisation (Simplification) des fonctions logiques . . . 15
4.1 Cellules universelles . . . 15
4.2 Simplification algébrique d'une fonction . . . 16
4.3 Simplification graphique : Tableaux de Karnaugh . . . 16
5 Représentation des fonctions logiques . . . 19
5.1 Logigramme . . . 19

Ce document évolue. Merci de signaler toutes erreurs ou coquilles.

1 Numération et codage

Le système de numérotation le plus utilisé aujourd'hui et le système décimal. Les systèmes automatiques utilisent plus naturellement le système binaire. La connaissance du système binaire et des changements de base binaire vers décimal et décimal vers binaire est donc essentielle à l'étude et à la réalisation des parties commandes.

1.1 Système de numération

1.1.1 Bases

Un nombre est représenté par la juxtaposition de symboles appelés digits pris parmi les éléments de la base $\mathscr{B}$ considérée.
Les vases les plus couramment utilisées sont :
- la base 10 (ou décimale) contenant 10 symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ;
- la base 2 (ou binaire) contenant 2 symboles : 0, 1 ;
- la base 16 (ou hexadécimale) comprenant 16 symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Base décimale $\mathscr{B}_{10}$	Base binaire $\mathscr{B}_2$	Base hexadécimale $\mathscr{B}_{16}$
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

1.1.2 Changement de base : transcodage

L'objectif est de mettre en place la correspondance entre l'expression d'un nombre dans une base et son expression dans une autre base.

Méthode

Passage d'une base X vers la base décimale

Notons a, b, c et d 4 digits et X une base quelconque. Pour obtenir $(abcd)_X$ dans la base décimale, il faut d'abord connaître la correspondance entre les digits des bases X et 10. La méthode est alors la suivante :

$$(abcd)_X = a_{10}\cdot X^3 + b_{10}\cdot X^2 + c_{10}\cdot X^1 + d_{10}\cdot X^0$$

On parle de système de numération pondéré. On appelle poids le rang de chaque digit. (a est le digit de poids le plus fort).

Exemple

Passer les nombres suivants en base décimale : $(1011)_2$ et $(A3B)_{16}$.

Méthode

Passage d'une base décimale vers la base X

On utilise la méthode des divisions successives. Soit un nombre N défini sur une base X. Son écriture en base 10 est la suivante :

$$(N)_X = n_p \cdot x^p + n_{p-1} \cdot x^{p-1} + ... + n_1 \cdot x^1 + n_0 \cdot x^0$$

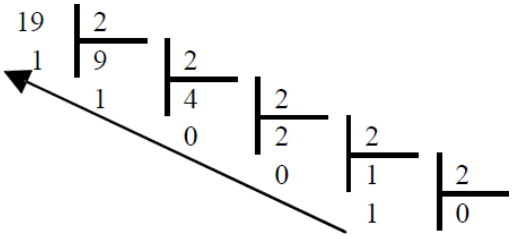
Effectuer le changement de base revient à déterminer les coefficients n_i.

En mettant x en facteur il vient :

$$(N)_X = x \cdot \left(n_p \cdot x^{p-1} + n_{p-1} \cdot x^{p-2} + ... + n_1 \cdot x^0 \right) + n_0$$

Cette expression est de la forme $(N)_X = x(\text{quotient}) + \text{reste}$. Le reste permet donc de déterminer n_0.

En reprenant le quotient précédent et en factorisant par x on trouve un nouveau reste n_1 et ainsi de suite jusqu'à n_p.
Ainsi pour écrire un nombre décimal en binaire, il faut faire des divisions successives par 2. Par exemple pour écrire 19 en binaire, on réalise les divisions ci-contre.
Le sens de lecture, et donc d'écriture se fait selon la flèche. On a donc $(19)_{10} = 1\,0011$.



Exemple

Écrire 25 en binaire et en hexadécimal.

1.2 Codage de l'information

Les ordinateurs travaillant en binaire, tout traitement informatique ou automatique nécessite de coder l'information en binaire à l'aide de deux états 1 ou 0. Un bit (contraction de binary digit) est un chiffre binaire (1 ou 0). un octet (byte en anglais) correspond à 8 bits. Un ko (kilo octet) correspond à 2^{10} chiffres soit 1024 octets.

L'écriture des nombres dans cette base donnant une multitude de chiffres, les informaticiens lui préfèrent le système hexadécimal qui permet de contracter les informations en paquets de 4 bits.

Il est également intéressant de regrouper un ensemble de valeurs binaires suivant une autre organisation qu'un système de nombres. Cette organisation sont appelées codes. Il en existe un très grand nombre et chacun peut en créer selon un besoin spécifique.

Les qualités requises pour un code sont principalement :
- la taille du codage (nombre de bits nécessaires) ;
- la fiabilité de lecture ;
- la simplification de manipulation.

Exemple

Sur une carte vitale, les données sont codées selon la norme ASCII (American Standard Code For Information Interchange) qui nécessite 8 bits et permet de coder 256 caractères. Les chiffre de 0 à 9 en ASCII binaire sont obtenus en ajoutant 0011 devant le code binaire du chiffre sur 4 bits. Les lettres de A à Z sont codées en ajoutant 010 devant le code binaire des 26 lettres (A=01000001, *etc*).

Exemple

Écrire la séquence suivante en ASCII binaire : 19 ans. Combien d'octets sont nécessaires pour coder cette séquence ?

On étudie dans la suite les principaux codes à connaître. On distingue les codes pondérés pour lesquels chaque chiffre possède un poids, des codes non pondérés qui nécessitent une table de correspondance pour décoder un nombre.

1.2.1 Code binaire naturel

Ce code pondéré correspond à donner un nombre selon sa valeur en système de numérotation binaire.

C'est le seul code qui permet de réaliser des opérations.

Inconvénient : il introduit des erreurs lors d'un changement de code. Pour passer de $(01)_2$ à $(10)_2$, donc de 1 à2, il faut modifier chaque digit et afficher (même brièvement) $(11)_2$ (soit 3) ou $(00)_2$ (soit 0). Cela peut provoquer des perturbations ou des erreurs.

Entrée				Sortie
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Table de vérité du code binaire naturel

1.2.2 Code décimal codé binaire (DCB)

C'est un code pondéré base sur le code binaire naturel mais qui est adapté à la représentation des nombres en base 10. En effet le code binaire pur n'associe pas des bits spécifiques aux unités, dizaines, centaines, ... La propriété du code DCB est d'associer 4 bits différents à chaque puissance de 10. Ainsi,

$$(1664)_{10} \quad \Longrightarrow \quad (0001.0110.0110.0100)_{DCB}$$

Ce code est utilisé pour les afficheurs 7 segments. Chaque afficheur reçoit le chiffre codé en binaire sur 4 bits.

Inconvénient : Cette représentation adaptée à la représentation binaire des nombres décimaux utilise un nombre de bits supérieur à celui du binaire naturel, et donc une place plus important en mémoire de l'ordinateur.

Remarque

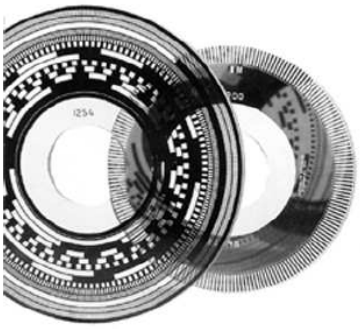
On peut réaliser la même typologie pour le code hexadécimal. Notons ce codage HCB. Ainsi,

$$(0C3F)_{16} \quad \Longrightarrow \quad (0000.1100.0011.1111)_{HCB}$$

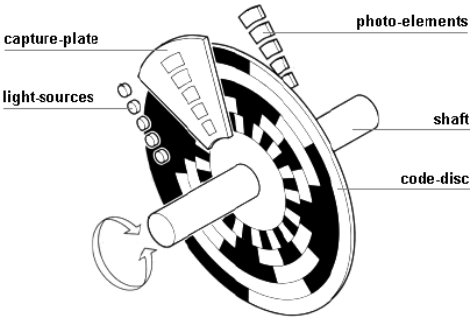
1.2.3 Code binaire réfléchi ou code Gray

Ce code non pondéré est un arrangement du système binaire. Le passage d'un nombre à l'autre se fait en changeant l'état d'un seul bit. Il est très utilisé pour décrire des automatismes (un changement d'état d'un composant correspond à un bit qui change), en particulier dans les codeurs de position absolue.

Avantage : il apporte une garantie d'interprétation avec une erreur maximale d'incrémentation.



Disques codeurs absolus et incrémentaux



Principe du codeur absolu

Pour obtenir le code Gray, il faut faire toutes les 2^1, 2^2, 2^3 ... lignes, une symétrie en commençant par le bit de droite et changer la valeur du bit de gauche.

Décimal	Binaire pur	Binaire réfléchi	
0	0 0 0 0	0 0 0 0	
1	0 0 0 1	0 0 0 1	Première symétrie
2	0 0 1 0	0 0 1 1	
3	0 0 1 1	0 0 1 0	Deuxième symétrie
4	0 1 0 0	0 1 1 0	
5	0 1 0 1	0 1 1 1	
6	0 1 1 0	0 1 0 1	
7	0 1 1 1	0 1 0 0	Troisième symétrie
8	1 0 0 0	1 1 0 0	
9	1 0 0 1	1 1 0 1	
10	1 0 1 0	1 1 1 1	
11	1 0 1 1	1 1 1 0	
12	1 1 0 0	1 0 1 0	
13	1 1 0 1	1 0 1 1	
14	1 1 1 0	1 0 0 1	
15	1 1 1 1	1 0 0 0	

2 Systèmes combinatoires

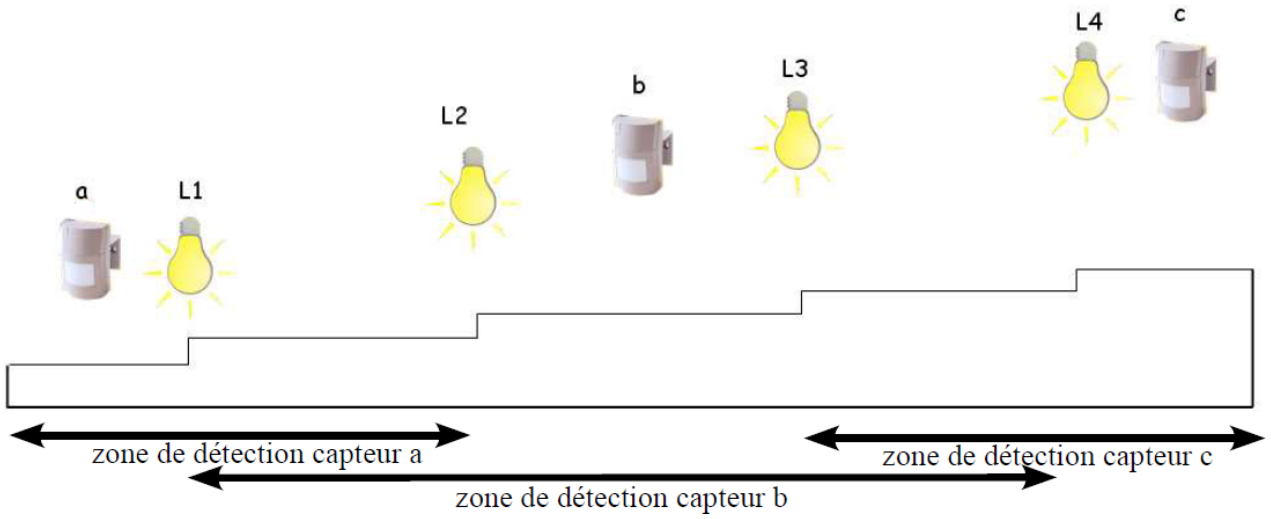
Exemple

Support : éclairage d'un escalier

On souhaite éclairer un escalier par 4 lampes qui s'allument automatiquement lorsque l'on monte ou descend l'escalier. Pour l'esthétisme, on désire que les lampes s'allument puis s'éteignent successivement lorsqu'une présence est détectée. On utilise 3 détecteurs de présence notés a, b, et c. L'objectif est de

Exemple

déterminer les équations logiques liant l'état des lampes à l'état des détecteurs de présence. Chaque capteur détecte une présence. Chaque capteur détecte une présence dans la zone indiquée. Il faut toujours 2 lampes d'allumées sauf aux marches extrêmes.



2.1 Définitions

2.1.1 Variables binaires

De nombreux composants utilisés en automatisme ne peuvent normalement prendre que deux états différents : lampe allumée ou éteinte, bouton-poussoir actionné ou relâché, moteur tournant ou à l'arrêt, vérin pneumatique sorti ou rentré.

A chacun de ces composants, on peut associer une variable binaire (ou logique, Tout Ou Rien) qui ne peut prendre que deux valeurs notées 0 ou 1 (vrai ou faux, oui ou non).

Définition

Une variable *a* est binaire si et seulement si elle peut prendre, à chaque instant, qu'une seule valeur parmi un ensemble de 2 valeurs possibles.

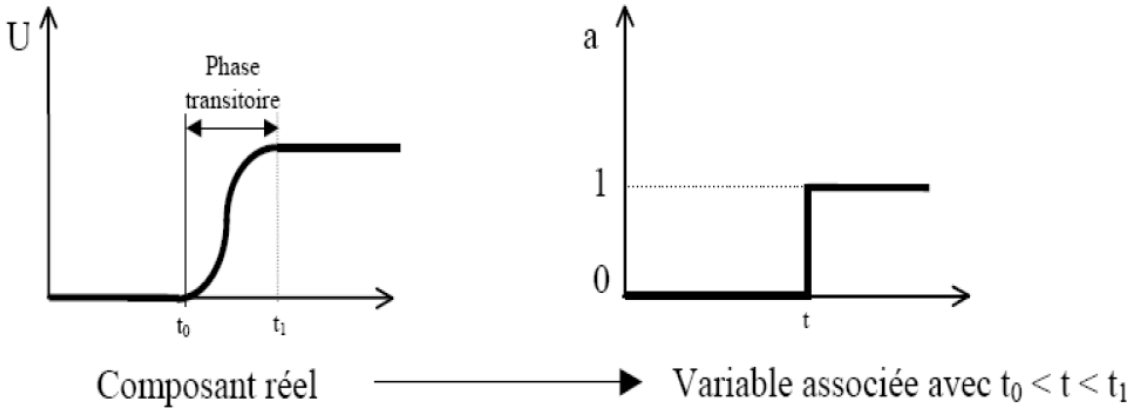
Exemple

Lister les variables binaires pour l'éclairage de l'escalier

Remarque

Le comportement tout ou rien (TOR) ne correspond qu'au comportement normalement prévu en régime stabilisé et en l'absence de tout dysfonctionnement.

L'association d'une variable binaire à un composant ne peut pas rendre compte des états transitoires apparaissent entre deux états stables. C'est donc une simplification du comportement réel.



2.1.2 Système binaire

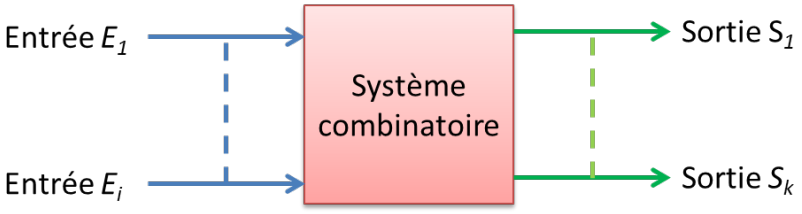
Définition

Un système est dit binaire ou logique si les variables d'entrée et de sortie sont binaires.

2.1.3 Systèmes combinatoire et séquentiel

Définition

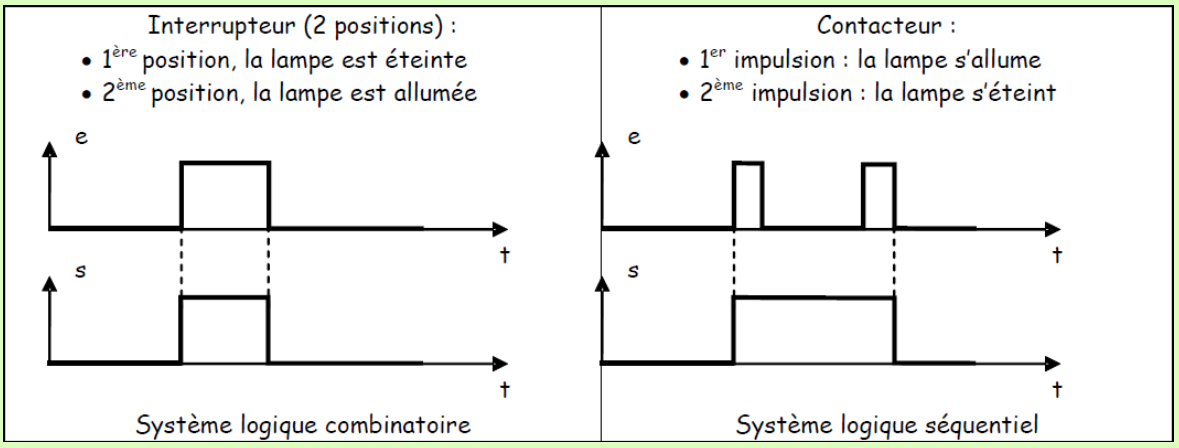
Un système logique combinatoire est un système binaire pour lequel à un état des variables d'entrée E_i correspond un unique état des variables de sortie S_j. (La réciproque n'est pas vraie.)



Définition

Un système logique est dit séquentiel si les sorties S_j ne dépendent pas uniquement des E_i.

Exemple



2.2 Algèbre de Boole

La fonction logique qui caractérise le système est indépendante du temps. Pour traiter de tels systèmes, on utilise l'algèbre de Boole.

2.2.1 Définition

C'est un algèbre de propositions logiques mise au point par un mathématicien anglais, Georges Boole (1815 – 1864).

Définition

Un ensemble E a une structure d'algèbre Boole si on a défini dans cet ensemble :
- une relation d'équivalence notée $=$;
- deux lois de composition interne $+$ (addition booléenne) et $\cdot$ (multiplication booléenne) ;
- une loi appelée complémentation ($\overline{a}$ complément de a).

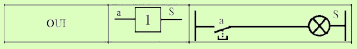
Une algèbre binaire est une algèbre de Boole dont les éléments B ne peuvent prendre que deux valeurs notées 0 ou 1 : $B = \{0,1\}$.

Exemple

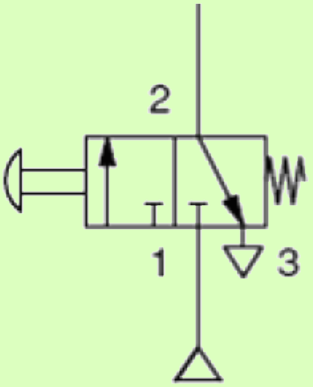
Fonction OUI.

Exemple

Fonction OUI – Table de vérité et logigramme



Fonction OUI en technologie électrique



Fonction OUI en technologie pneumatique

2.2.2 Fonction NON – appelée complément

Définition

Fonction NON – appelée complément

$$B \longmapsto B$$
$$a \longmapsto \overline{a}$$

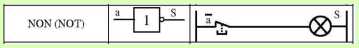
Il faut lire $\overline{a}$: NON a.

a	$\overline{a}$
0	1
1	0

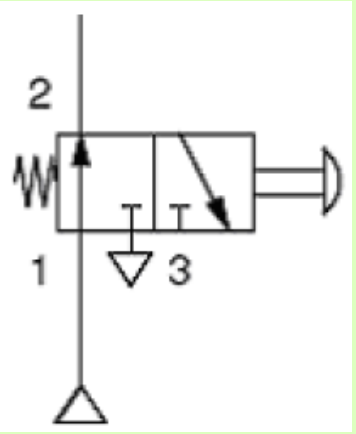
Exemple

Fonction NON.

Fonction NON – Table de vérité et logigramme



Fonction NON en technologie électrique



Fonction NON en technologie pneumatique

2.2.3 Fonction ET – Produit booléen

Définition

Fonction ET – Produit booléen

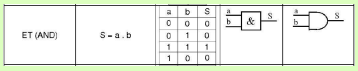
$$\begin{array}{rcl} B \times B & \longmapsto & B \\ (a,b) & \longmapsto & a \cdot b \end{array}$$

Il faut lire a ET b.

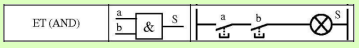
a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Exemple

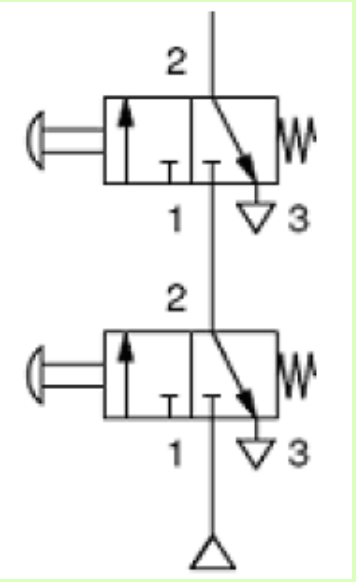
Fonction ET.



Fonction ET – Table de vérité et logigramme



Fonction ET en technologie électrique



Fonction ET en technologie pneumatique

2.2.4 Fonction OU – Somme booléenne

Définition

Fonction OU – Somme booléenne

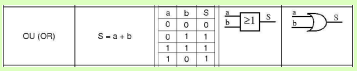
$$\begin{array}{rcl} B \times B & \longmapsto & B \\ (a,b) & \longmapsto & a+b \end{array}$$

Il faut lire a OU b.

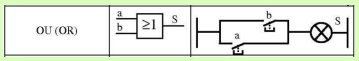
a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1

Exemple

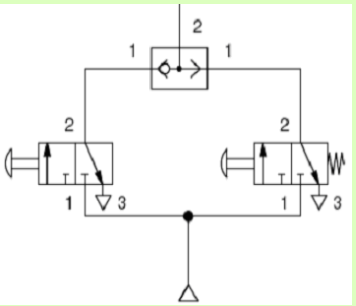
Fonction OU.



Fonction OU – Table de vérité et logigramme



Fonction OU en technologie électrique



Fonction OU en technologie pneumatique

2.2.5 Propriétés des opérateurs de base de l'algèbre de Boole

Propriété

Commutativité

$$a+b=b+a \qquad a\cdot b=b\cdot a$$

Propriété

Distributivité

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

Propriété

Associativité

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

Propriété

Élément neutre

$$a+0=a \qquad a\cdot 1=a$$

Propriété

Élément absorbant

$$a+1=1 \qquad a\cdot 0=0$$

Propriété

Complémentarité

$$a+\overline{a}=1 \qquad a\cdot\overline{a}=0$$

Par ailleurs $\overline{\overline{a}}=a$.

Propriété

Idem potence

$$a+a=a \qquad a\cdot a=a$$

Propriété

Identités remarquables

Absorption :

$$a+a\cdot b=a$$

Inclusion :

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

2.2.6 Théorème de Morgan

Théorème

$$\overline{\sum_{i=1}^{n} a_i}=\prod_{i=1}^{n}\overline{a_i}$$

$$\overline{\prod_{i=1}^{n} a_i}=\sum_{i=1}^{n}\overline{a_i}$$

Exemple

Déterminer les expressions complémentaires $\overline{P}$ et $\overline{Q}$ des expressions suivantes : $P=x\cdot y\cdot(z+\overline{t})$ et $Q=\overline{x}\cdot\overline{y}+y+x\cdot t$.

3 Fonctions logiques

Un système combinatoire est décrit par une fonction logique qui permet de définir de manière unique la sortie pour une combinaison des entrées.

3.1 Table de vérité

La table de vérité d'une fonction logique est une écriture systématique qui consiste à décrire toutes les combinaison des variables et à y associer les valeurs correspondantes de la fonction.

Exemple

La table de vérité pour les lampes L_1, L_2, L_3, L_4 de l'éclairage d'escalier est la suivante :

a	b	c	L_1	L_2	L_3	L_4
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	1
0	1	0	0	1	1	0
1	1	0	1	1	0	0
1	1	1	1	1	1	1
1	0	1	1	0	0	1
1	0	0	1	0	0	0

Compléter la table de vérité suivante et vérifier les théorèmes de De Morgan pour deux variables.

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

Il est possible à l'aide de la table de vérité de définir une équation donnant les variables de sortie en fonction des variables d'entrée. Il existe 2 modes d'écriture particuliers :

- somme canonique (somme de produit) : $S = a \cdot b + c \cdot d$;
- produit canonique (produit de sommes) : $S = (a+b) \cdot (c+d)$.

Méthode

Détermination de la somme canonique

Pour chaque ligne où la sortie vaut 1, déterminer la combinaison d'entrées correspondante à l'aide de l'opérateur ET puis sommer ces combinaisons.

Exemple

Donner l'expression sous forme canonique de L_1 et L_2 à partir de la table de vérité précédente.

Remarque

On utilise la forme de sommes canoniques si le nombre de 1 est inférieur au nombre de 0. Il est souvent nécessaire de simplifier ces équations pour réaliser technologiquement ces fonctions (voir plus loin).

Méthode

Détermination du produit canonique

Déterminer l'expression de $\overline{S}$ ce qui revient à : Déterminer la combinaison d'entrées pour chaque ligne où la sortie vaut 0, les sommer, puis passez au complémentaire en utilisant les théorèmes de De Morgan.

Exemple

Donner l'expression sous forme de produit canonique de L_1 et L_2.

Remarque

Cette méthode est utile lorsque la sortie comporte peu de 0 et beaucoup de 1 ou lorsque l'on recherche la fonction complémentaire.

4 Réorganisation (Simplification) des fonctions logiques

La simplification des expressions logiques est destinée à économiser le matériel nécessaire à la réalisation (utilisation d'un composant réalisant plusieurs fonctions identiques) ou diminuer l'importance des équations programmées.

4.1 Cellules universelles

Définition

Une cellule (ou fonction) est dite "universelle" si elle permet de réaliser les fonctions ET, OU, NON. Il est alors possible de réaliser toutes les fonctions logiques à l'aide de cette seule cellule.

Exemple

Les cellules NAND (non et – $aNANDb = \overline{a \cdot b}$) et NOR (non ou – $aNORb = \overline{a+b}$) sont des cellules universelles.

Réalisation de la fonction NON : $NON(a) = \overline{a} = \overline{a \cdot a}$. Besoin d'une seule cellule NAND.

Réalisation de la fonction ET : $aETb = a \cdot b = \overline{\overline{a \cdot b}}$. Besoin de deux cellules NAND en cascade.

Réalisation de la fonction OU : $aOUb = a+b = \overline{\overline{a+b}} = \overline{\overline{a} \cdot \overline{b}}$. Besoin de trois cellules NAND en cascade.

De même, la cellule NOR est universelle.

Exemple

Écrire l'expression logique suivante uniquement avec des opérateurs NAND puis indiquer le nombre minimal d'opérateurs utilisés.

$$F = c \cdot (\overline{a} + \overline{b})$$

4.2 Simplification algébrique d'une fonction

Il s'agit d'utiliser les propriétés, théorèmes et identités remarquables de l'algèbre de Boole afin de simplifier une équation. Cette méthode n'est pas forcément simple à utiliser et demande beaucoup d'intuition.

Exemple

$$F(a,b,c) = \overline{a}\cdot\overline{b}\cdot c + a\cdot\overline{b}\cdot c + \overline{a}\cdot b\cdot c = \overline{a}\cdot c\cdot(\overline{b}+b) + a\cdot\overline{b}\cdot c = \overline{a}\cdot c + a\cdot\overline{b}\cdot c = c(\overline{a}+a\cdot\overline{b}) = c\left(\overline{a}(1+\overline{b}) + a\cdot\overline{b}\right) = c\cdot(\overline{a}+\overline{b})$$

Exemple

Simplifier par cette méthode l'expression des lampes L_1 et L_2.

4.3 Simplification graphique : Tableaux de Karnaugh

Le tableau de Karnaugh est une présentation particulière de la table de vérité permettant de simplifier graphiquement une expression analytique.

4.3.1 Construction du tableau de Karnaugh

Un tableau de Karnaugh comporte autant d'entrées qu'il y a de variables et associe une case à chaque combinaison de ces variables. Pour une fonction de n variables, un tableau de Karnaugh comporte donc 2^n cases. Le tableau répartit les entrées en ligne et en colonne de manière à n'obtenir qu'un seul changement d'une entrée au passage d'un case à une case mitoyenne (code binaire réfléchi obtenu par symétries). Ainsi pour une fonction à n variables, le passage d'une case adjacente à une autre correspond au changement de valeur d'une seule variable.

a	0	1
	F(0)	F(1)

Tableau à 1 variable

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

Tableau à 2 variables

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

Tableau à 3 variables

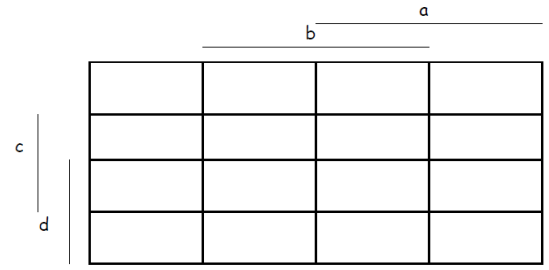


Tableau à 4 variables

Attention : Le trait permet de simplifier l'écriture et correspond à la valeur logique 1 de chaque variable

Tableau à 4 variables

Exemple

Écrire le tableau de Karnaugh des lampes L_1 et L_2.

4.3.2 Simplification par tableau de Karnaugh

Les simplifications d'une fonction à l'aide de son tableau de Karnaugh consistent à faire des regroupements de 1 adjacent par puissance de deux. Ces regroupements doivent être les plus gros possibles et en nombre minimum afin d'obtenir l'expression la plus simple possible. Des recoupements entre les regroupements sont possibles.

Pour que la simplification ait lieu, les regroupements doivent être disposés de manière symétrique par rapport aux axes de symétrie du tableau. Ces axes correspondent aux symétries effectuées pour établir le code Gray.

4.3.3 Exemple détaillé

Considérons la fonction $F(a,b,c)=\overline{a}\cdot\overline{b}\cdot c+a\cdot\overline{b}\cdot c+\overline{a}\cdot b\cdot c$ étudiée précédemment, décrite par sa table de vérité et établissons son tableau de Karnaugh.

Un premier regroupement de 1 apparaît facilement. Dans ce regroupement, la variable b intervient sous forme de 1 et de 0 et peut être ignorée.

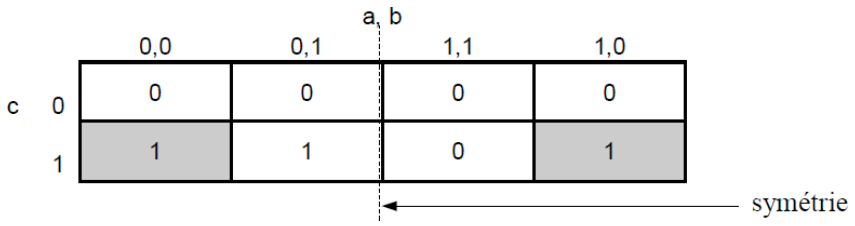
c	a, b = 0,0	0,1	1,1	1,0
0	0	0	0	0
1	1	1	0	1

Ce regroupement est équivalent à $\overline{a}\cdot c$.

En effet, nous avons pour ces deux cases : $(\overline{a}\cdot\overline{b}\cdot c)+(\overline{a}\cdot b\cdot c)=\overline{a}\cdot c(\overline{b}+b)=\overline{a}\cdot c$.

Le résultat fait normalement intervenir deux variables parce qu'il y a trois variables d'entrée et que la simplification s'est effectuée au moyen d'un regroupement de 2^1 cases.

Un deuxième regroupement peut être effectué si on prend en compte la symétrie verticale (ceci correspond dans le cadre des tableaux à 3 ou 4 variables à considérer les colonnes ou les lignes extrêmes adjacentes).



Ce regroupement est équivalent à $\overline{b}\cdot c$.

En effet, nous avons pour ces deux cases : $(\overline{a}\cdot\overline{b}\cdot c)+(a\cdot\overline{b}\cdot c)=\overline{b}\cdot c(\overline{a}\cdot a)=\overline{b}\cdot c$. Le résultat fait normalement intervenir deux variables parce qu'il y a trois variables d'entrée et que la simplification s'est effectuée au moyen d'un groupement de 2^1 cases. La fonction $F(a,b,c)$ s'écrit donc : $F(a,b,c)=\overline{a}\cdot c+\overline{b}\cdot c=(\overline{a}+\overline{b})\cdot c$.

Remarque

Il est possible de faire des regroupements de 0 pour obtenir la fonction complémentée.

Exemple

Simplifier l'expression des lampes L_1 et L_2 à partir des tableaux de Karnaugh.

4.3.4 Cas des fonctions incomplètes

Le cas des fonctions incomplètes (ou incomplètement spécifiées) peut apparaître comme un cas particulier. En réalité, on peut considérer qu'en pratique c'est le cas le plus fréquent. En effet :

- soit la valeur de la fonction n'a pas réellement d'importance pour le problème considéré ;
- soit certains combinaisons de variables sont physiquement impossibles.

Dans ces conditions, la case correspondant à la combinaison non spécifiée sera marquée par un symbole Φ ou laissée vide.

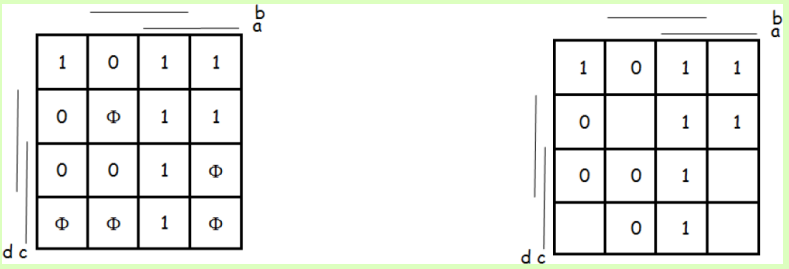
La démarche de simplification d'une telle fonction se déroule de la même manière que précédemment. Il faut choisir des regroupements de 2^1, 2^2, ... cases. Si le regroupement se fait par les 1 (respectivement 0) alors les cases contenant Φ qui interviennent dans le regroupement prenant la valeur 1 (respectivement 0), et les cases ne contenant pas Φ qui interviennent dans le regroupement prennent la valeur 0 (respectivement 1). La fonction est ainsi complètement spécifiée.

Remarque

Il n'est pas nécessaire de faire intervenir tous les Φ dans les regroupements.

Exemple

Simplifier le tableau suivant.



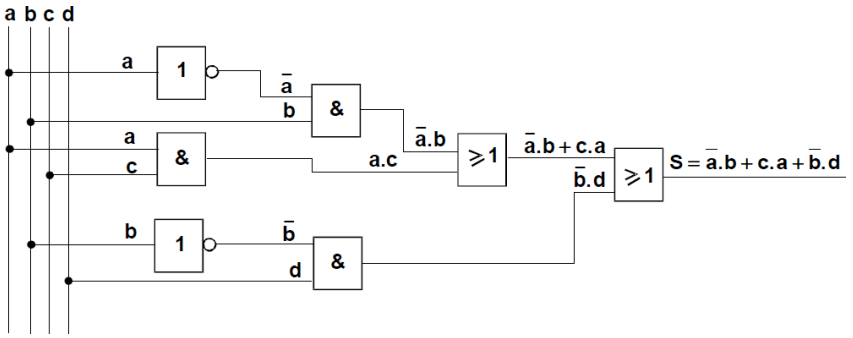
5 Représentation des fonctions logiques

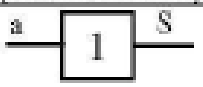
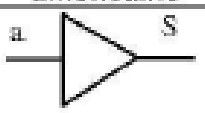
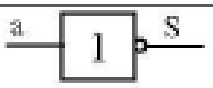
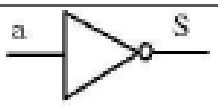
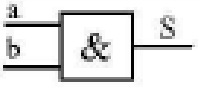
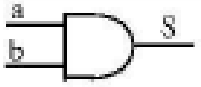
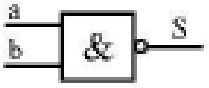
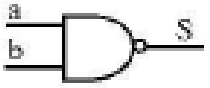
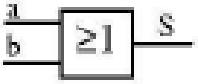
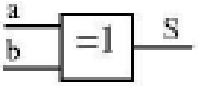
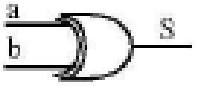
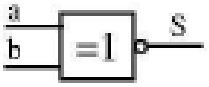
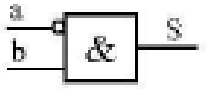
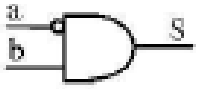
Dans un système combinatoire :

- les variables d'entrée sont des informations qui décrivent à un moment donnée l'état de certaines parties (position d'un chariot, d'un outil, ...)
- les variables de sortie permettent de décrire l'état dans lequel doivent se retrouver les organes récepteurs.

Une fois l'expression logique obtenue et simplifiée, il est possible d'exprimer graphiquement le comportement du système combinatoire. On utilise pour cela en général le logigramme ou le schéma électrique. On peut également utiliser un schéma pneumatique ou électronique.

5.1 Logigramme



Opérateur	Fonction logique	Table de vérité	Norme IEC (internationale)	Norme américaine
OUI	$S = a$	a | S 0 | 0 1 | 1		
NON (NOT)	$S = \overline{a}$	a | S 0 | 1 1 | 0		
ET (AND)	$S = a \cdot b$	a b | S 0 0 | 0 0 1 | 0 1 1 | 1 1 0 | 0		
NON-ET (NAND))	$S = \overline{a.b} = \overline{a}+\overline{b}$	a b | S 0 0 | 1 0 1 | 1 1 1 | 0 1 0 | 1		
OU (OR)	$S = a + b$	a b | S 0 0 | 0 0 1 | 1 1 1 | 1 1 0 | 1		
NON-OU (NOR)	$S = \overline{a+b} = \overline{a}.\overline{b}$	a b | S 0 0 | 1 0 1 | 0 1 1 | 0 1 0 | 0		
OU-exclusif (XOR)	$S = a \oplus b = \overline{a}b + a\overline{b}$	a b | S 0 0 | 0 0 1 | 1 1 1 | 0 1 0 | 1		
ET-exclusif (XAND)	$S = \overline{a}\overline{b} + ab = \overline{a \oplus b}$	a b | S 0 0 | 1 0 1 | 0 1 1 | 1 1 0 | 0		
INHIBITION	$S = \overline{a}b$	a b | S 0 0 | 0 0 1 | 1 1 0 | 0		

Références

[1] Supports de cours de David Violeau, Lycée Saint-Louis, Paris

[2] Supports de cours de Florestan Mathurin, Lycée Bellevue, Toulouse