Codage de l'information et systèmes logiques

Cours

Programme - Compétences		
B216	MODELISER	Systèmes logiques: - codage de l'information - binaire naturel, binaire réfléchi - représentation hexadécimale - table de vérité - opérateurs logiques fondamentaux (ET, OU, NON)

A. Logique combinatoire

A.I. Introduction - Définitions

Les **systèmes logiques combinatoires** sont des systèmes pour lesquels des causes combinées impliquent immédiatement des effets. A tout moment, les mêmes causes impliquent toujours les mêmes effets. Causes et effets sont des **propositions logiques** associés à des **états** « **vrai** » ou « **faux** » traduits par deux variables 1 (vrai) ou 0 (faux). Il est ainsi possible d'utiliser **l'algèbre de Boole** afin de représenter les relations entre causes et effets par des **fonctions logiques**. L'ensemble des relations dites **causales** (à une cause correspond un effet) constitue un « **modèle de fonctionnement logique** » d'un système.

La réalisation pratique des fonctions logiques permettant d'induire des causes connaissant les effets s'effectue par des moyens divers, les plus utilisés étant l'électrique, le pneumatique et l'hydraulique.

Un système logique peut être combinatoire ou séquentiel. Les systèmes combinatoires sont des systèmes dans lesquels une combinaison des entrées induit une et une seule sortie. Prenons l'exemple d'un portail automatique : si une action à un instant t sur une commande d'ouverture ou de fermeture induit une ouverture complète ou une fermeture complète après avoir relâché le bouton, on voit que pour un même état des entrées, il existe 2 situations possibles : ouverture ou fermeture. Ceci est un système séquentiel dans lequel il faudra un automate pour gérer les séquences des actions. Si par contre il faut nécessairement laisser un bouton appuyé pour fermer et un bouton appuyé pour ouvrir, le système est combinatoire et des fonctions logiques suffiront à elles seules pour le faire fonctionner.

A.II. Représentation d'un modèle de fonctionnement logique

Il existe différents outils permettant de représenter un modèle de fonctionnement logique d'un système. Traitons un exemple pour introduire ces outils.

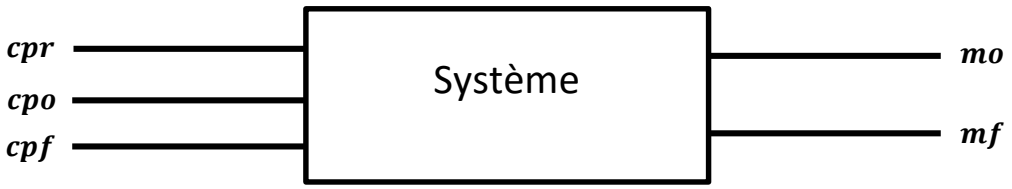
Soit une porte coulissante automatique gérée à l'aide

- d'un capteur de présence qui couvre en même temps les zones intérieure et extérieure (fonction de détection de présence d'un usager) et la zone de passage (fonction de sécurité) associé à la variable « cpr »
- d'un capteur « Porte ouverte » associé à la variable « cpo »
- d'un capteur « Porte fermée » associé à la variable « cpf »
- d'un moteur recevant deux ordres qui ne peuvent être envoyés simultanément
    - ordre d'ouverture associé à la variable « mo »
    - ordre de fermeture associé à la variable « mf »

Le modèle de fonctionnement logique de ce système doit permettre de représenter les ordres « mo » et « mf » à l'aide des diverses entrées issues des capteurs « cpr », « cpo » et « cpf ».

A.II.1 Représentation globale du système

Le système logique proposé se schématise en fonction de ses entrées et sorties ainsi :



A.II.2 Table de vérité

Le premier outil à notre disposition est la table de vérité. On réalise un tableau répertoriant

- dans sa partie gauche les entrées en colonne et l'intégralité des états possibles de celles-ci
- dans sa partie droite les sorties et les états qu'elles doivent prendre en fonction des combinaisons d'entrées existantes

Pour créer ce tableau :

- on met autant de colonnes que de variables, ici 5
- on met 2^n lignes, n correspondant au nombre d'entrées, ici 3, soit 8 lignes

Entrées			**Sorties**	
cpr	cpo	cpf	mo	mf

On remplit ensuite les colonnes associées aux entrées afin de représenter l'intégralité des combinaisons possibles des entrées. On peut par exemple utiliser le code binaire :

Entrées			**Sorties**	
cpr	cpo	cpf	mo	mf
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Détaillons les différents cas rencontrés et voyons si la table de vérité peut être remplie directement.

cpr	cpo	cpf	Etat porte	Présence usager	Action
0	0	0	Ni ouverte - Ni fermée	Non	Fermer
0	0	1	Fermée		Ne rien faire
0	1	0	Ouverte		Fermer
0	1	1	Ouverte et fermée		*Etat impossible*
1	0	0	Ni ouverte - Ni fermée	Oui	Ouvrir
1	0	1	Fermée		Ouvrir
1	1	0	ouverte		Ne rien faire
1	1	1	Ouverte et fermée		*Etat impossible*

On voit apparaître deux états impossibles. Dans le cas **d'états impossibles**, ou **d'états indéterminés** (le cahier des charges ne précise pas les actions à mener dans des cas particuliers), ce sera au concepteur de choisir

- Pour des états pouvant mettre en danger la sécurité des usagers, le concepteur devra choisir la solution la plus appropriée. Ici, si l'un des capteurs « ouvert » ou « fermé » est défaillant et que les informations « ouvert » et « fermé » arrivent en même temps, il sera préférable d'imposer une ouverture (ou ne rien faire) afin de ne pas risquer de fermer la porte sur un usager, en particulier dans le cas de la dernière ligne du tableau.
- Pour des états indéterminés, le concepteur choisira la solution qui l'arrange. Il arrive souvent que ce choix permette de simplifier la fonction logique finale.

Nous pouvons donc choisir les actions à mener en fonction des entrées :

cpr	cpo	cpf	Action
0	0	0	Fermer
0	0	1	Ne rien faire
0	1	0	Fermer
0	1	1	*Ouvrir*
1	0	0	Ouvrir
1	0	1	Ouvrir
1	1	0	Ne rien faire
1	1	1	*Ouvrir*

Voici la table de vérité représentant le modèle de fonctionnement logique du système étudié.

Entrées			**Sorties**	
cpr	cpo	cpf	mo	mf
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

Table de vérité du système étudié

A.II.3 Fonctions logique

Les fonctions logiques permettent de représenter les variables de sortie « mo » et « mf » en fonction des entrées à l'aide des outils de l'algèbre de Boole.

Entrées			**Sorties**	
cpr	cpo	cpf	mo	mf
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

En utilisant la table de vérité, on montre les relations suivantes :

$$\begin{cases} mo = \overline{cpr}.cpo.cpf + cpr.\overline{cpo}.\overline{cpf} + cpr.\overline{cpo}.cpf + cpr.cpo.cpf \\ mf = \overline{cpr}.\overline{cpo}.\overline{cpf} + \overline{cpr}.cpo.\overline{cpf} \end{cases}$$

Cette forme non simplifier s'appelle la **première forme canonique** des expressions de mo et mf.

Il suffit alors d'utiliser les outils de l'algèbre de Boole afin de simplifier ces expressions :

$$mo = \overline{cpr}.cpo.cpf + cpr.\overline{cpo}.\overline{cpf} + cpr.\overline{cpo}.cpf + cpr.cpo.cpf$$

$$= cpo.cpf.(cpr + \overline{cpr}) + cpr.\overline{cpo}.\left(\overline{cpf} + cpf\right)$$

$$= cpo.cpf + cpr.\overline{cpo}$$

$$mf = \overline{cpr}.\overline{cpo}.\overline{cpf} + \overline{cpr}.cpo.\overline{cpf} = \overline{cpr}.\overline{cpf}(\overline{cpo} + cpo) = \overline{cpr}.\overline{cpf}$$

Soit finalement :

$$\begin{cases} mo = cpo.cpf + cpr.\overline{cpo} \\ mf = \overline{cpr}.\overline{cpf} \end{cases}$$

Fonctions logiques du système étudié

Remarques :

- cette étape est très importante car plus la simplification sera bien faite, plus le logigramme et le schéma électrique du câblage du système seront concis et simples.
- La simplification de ces expressions s'effectue simplement à l'aide des tableaux de Karnaugh qui ne sont plus au programme

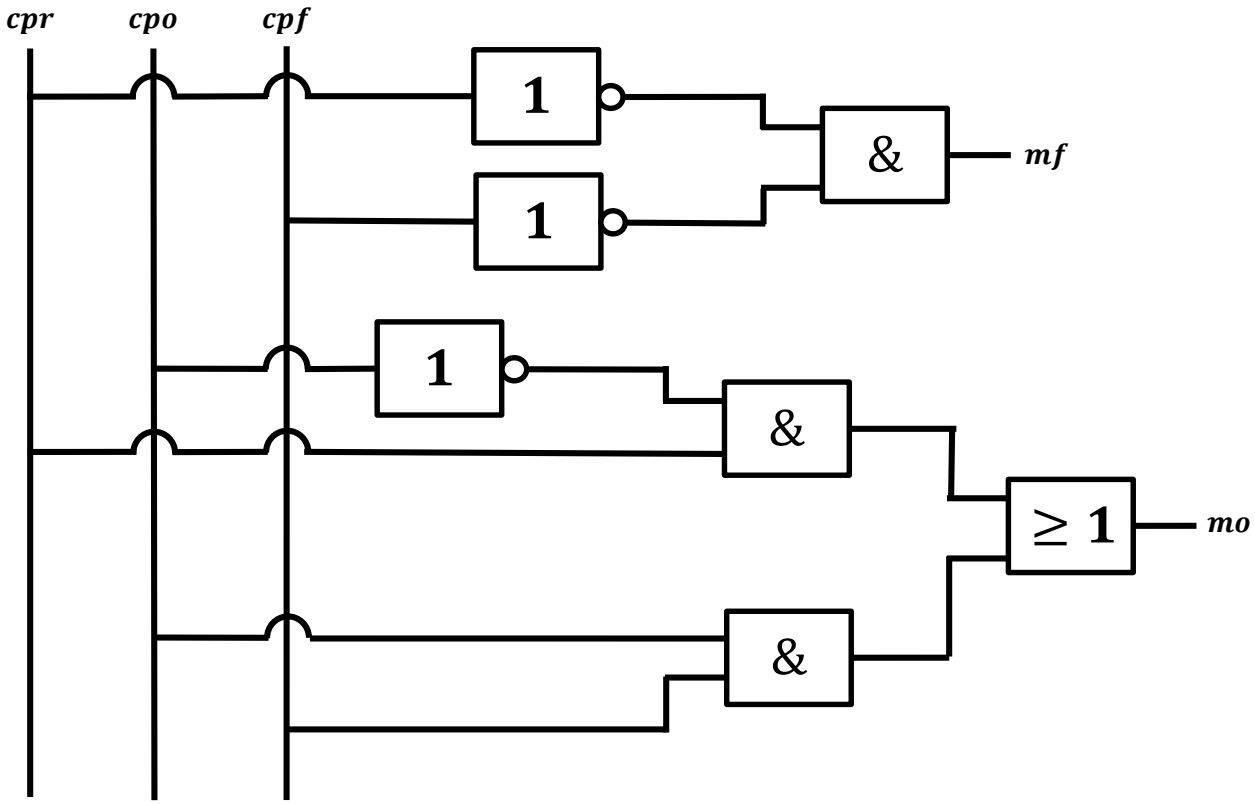
A.II.4 Logigramme

Un logigramme est une représentation quasi fonctionnelle du câblage du système.

On place dans sa partie gauche autant de traits verticaux qu'il y a d'entrées. Ce sont comme des fils électriques comportant la valeur 1 associée à la variable concernée.

Il suffit alors d'utiliser les opérateurs logiques à notre disposition dans le but de d'obtenir en sortie la valeur des variables de sortie en fonction des variables d'entrée en utilisant les fonctions logiques du système :

$$\begin{cases} mo = cpo.cpf + cpr.\overline{cpo} \\ mf = \overline{cpr}.\overline{cpf} \end{cases}$$



Logigramme
du système étudié

A.II.5 Schéma à contacts

On peut finalement représenter le modèle de fonctionnement logique du système étudié à l'aide du schéma à contacts, première étape vers la réalisation électrique du système.

Les quelques caractéristiques de réalisation des schémas à contacts à connaître sont les suivantes :

- La fonction ET est représentée par une mise en série des contacts
- La fonction OU est représentée par une mise en parallèle des contacts
- Une variable « normale » (a) est représentée par un contacteur normalement ouvert
- Une variable « opposée » $(\bar{a})$ est représentée par un contacteur normalement fermé
- La sortie est représentée par un cercle avec une croix à l'intérieur

On obtient ainsi pour notre système :

$$\begin{cases} mo = cpo.cpf + cpr.\overline{cpo} \\ mf = \overline{cpr}.\overline{cpf} \end{cases}$$

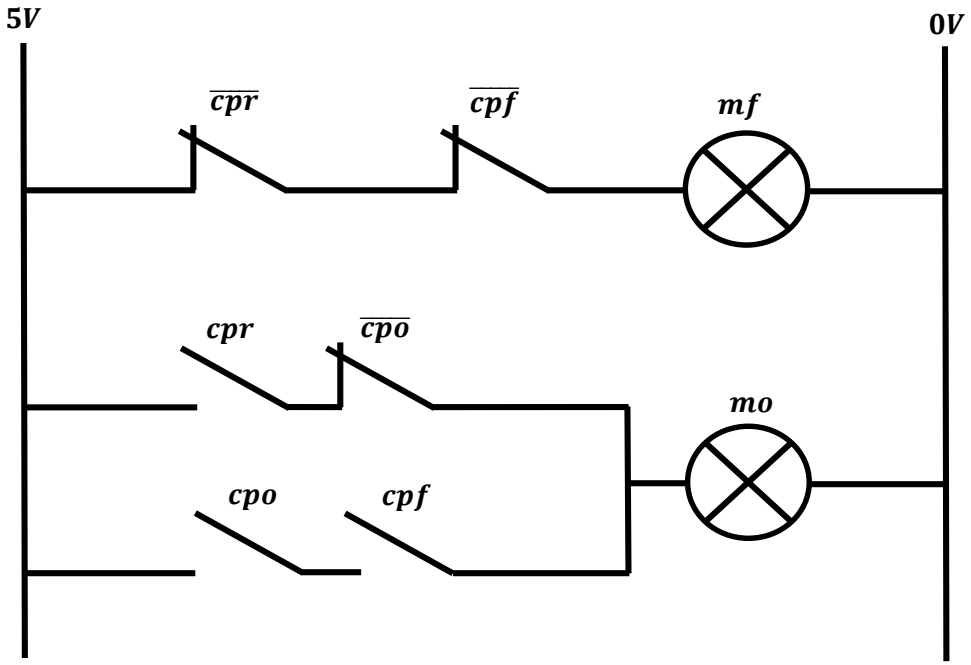


Schéma à contacts du système étudié

Remarque :

- Un schéma à contact représente le système lorsque toutes les variables sont à l'état 0
- Un changement d'une variable implique la fermeture du contact s'il est normalement ouvert et son ouverture s'il est normalement fermé

A.III. Algèbre de Boole

L'algèbre de Boole permet de manipuler des variables qui prennent des valeurs de 0 ou 1 afin de représenter des propositions logiques et de garder leur sens.

A.III.1 Propriétés

Soient a et b deux variables valant 0 ou 1.

Fonction ET	$a.b$
Fonction OU	$a+b$
Commutativité	$a.b=b.a$ $a+b=b+a$
Associativité	$(a.b).c=a.(b.c)$ $(a+b)+c=a+(b+c)$
Distributivité	$a.(b+c)=(a.b)+(a.c)$ $a+(b.c)=(a+b).(a+c)$
Eléments neutres	$a+0=a$ $a.0=0$ $a+1=1$ $a.1=a$
Identités remarquables	$a+\bar{a}.b=a+b$ $a+a.b=a$
Idempotence	$a+a=a$ $a.a=a$
Complémentation	$a+\bar{a}=1$ $a.\bar{a}=0$
Théorèmes de De Morgan	$\overline{a+b}=\bar{a}.\bar{b}$ $\overline{a.b}=\bar{a}+\bar{b}$
Ou exclusif	$\bar{a}.b+a.\bar{b}=a\oplus b$
Identité	$\bar{a}.\bar{b}+a.b=\overline{a\oplus b}$

Toutes ces propriétés doivent être maîtrisées dans le but de simplifier les expressions des fonctions logiques d'un système.

A.III.2 Simplification d'expression

Soit l'expression :

$$s = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.\bar{b}.c.d + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.\bar{c}.d$$

Le choix est volontairement fait de prendre une expression particulière qui conduit à deux expressions simplifiées différentes selon les regroupements effectués. Nous verrons alors une méthode permettant de trouver l'expression la plus simple.

$s = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.\bar{b}.c.d$ $+\bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.\bar{c}.d$	$s = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.\bar{c}.d$ $+\bar{a}.\bar{b}.c.\bar{d} + \bar{a}.\bar{b}.c.d$
$s = \bar{a}.\bar{b}.(\bar{c}.\bar{d} + \bar{c}.d + c.\bar{d} + c.d)$ $+\bar{a}.b.\bar{c}(\bar{d} + d)$ $s = \bar{a}.\bar{b}.\left(\bar{c}(\bar{d} + d) + c(\bar{d} + d)\right)$ $+\bar{a}.b.\bar{c}$ $s = \bar{a}.\bar{b}.(\bar{c} + c)$ $+\bar{a}.b.\bar{c}$ $s = \bar{a}.\bar{b} + \bar{a}.b.\bar{c}$	$s = \bar{a}.\bar{b}.\bar{c}.(\bar{d} + d) + \bar{a}.b.\bar{c}.(\bar{d} + d)$ $+\bar{a}.\bar{b}.c(\bar{d} + d)$ $s = \bar{a}.\bar{b}.\bar{c} + \bar{a}.b.\bar{c}$ $+\bar{a}.\bar{b}.c$ $s = \bar{a}.\bar{c}(\bar{b} + b)$ $+\bar{a}.\bar{b}.c$ $s = \bar{a}.\bar{c} + \bar{a}.\bar{b}.c$

A ce stade, on peut se demander laquelle est la meilleur. En réalité, **l'astuce** consiste à ajouter, si c'est utile, un ou plusieurs termes de l'égalité de départ permettant de simplifier encore l'expression obtenue :

$s = \bar{a}.\bar{b} + \bar{a}.b.\bar{c}$	$s = \bar{a}.\bar{c} + \bar{a}.\bar{b}.c$
« On voit » qu'un terme en $\bar{a}.\bar{b}.\bar{c}$ simplifierait grandement cette expression : $s = \bar{a}.\bar{b} + \bar{a}.b.\bar{c} + \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d$ Cette égalité reste vraie puisque les 2 termes ajoutés appartiennent à la solution. $s = \bar{a}.\bar{b} + \bar{a}.b.\bar{c} + \bar{a}.\bar{b}.\bar{c}$ $s = \bar{a}.\bar{b} + \bar{a}.\bar{c}$ $s = \bar{a}.(\bar{b} + \bar{c})$	« On voit » qu'un terme en $\bar{a}.\bar{b}.\bar{c}$ simplifierait grandement cette expression : $s = \bar{a}.\bar{c} + \bar{a}.\bar{b}.c + \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d$ Cette égalité reste vraie puisque les 2 termes ajoutés appartiennent à la solution. $s = \bar{a}.\bar{c} + \bar{a}.\bar{b}.c + \bar{a}.\bar{b}.\bar{c}$ $s = \bar{a}.\bar{c} + \bar{a}.\bar{b}$ $s = \bar{a}.(\bar{b} + \bar{c})$

Remarques :

- La présence de cas indéterminés dans un système peut permettre, en choisissant bien la réponse dans ces cas, de simplifier les fonctions logiques de la même façon
- L'utilisation des tableaux de Karnaugh qui n'est plus au programme simplifiait grandement le travail de simplification

A.III.3 Tableaux de Karnaugh – Hors programme

Le principe des tableaux de Karnaugh est simple :

Pour chaque variable de sortie s, on crée un tableau qui est fonction des entrées. Généralement, on a entre 2 et 4 entrées.

2 entrées

s		
$a\backslash b$	0	1
0		
1		

3 entrées

s		
$ab\backslash c$	0	1
00		
01		
11		
10		

4 entrées

s				
$ab\backslash cd$	00	01	11	10
00				
01				
11				
10				

Pour créer ces tableaux, on doit :

- Indiquer les variables de la colonne de gauche et de la ligne du haut (choix)
  - Variables a ou $a\&b$ pour la colonne de gauche
  - Variables b, c ou $c\&d$ pour la ligne du haut
- On indique ensuite les possibilités de valeurs de ces variables en prenant soin de faire changer 1 seul Bit à chaque changement de ligne ou colonne, même entre la première et la dernière colonne et la première et la dernière ligne (code Gray adapté)
- On remplit alors les cases avec des 0 et des 1 indiquant les valeurs de s pour chaque combinaison des entrées
- On réalise des paquets carrés ou rectangulaires contenant le plus possible de 1 afin de déterminer s (ou de 0 pour déterminer $\bar{s}$ puis s) dont le nombre de lignes et de colonnes est une puissance de 2, tout en sachant que le tableau est « infini », c'est-à-dire que l'on peut regrouper les termes de la colonne de gauche avec ceux de la colonne de droite, ou de la dernière ligne avec ceux de la première ligne, les 4 coins… Il faut au minimum prendre une fois chaque terme (les 1 pour s ou les 0 pour $\bar{s}$)
- Finalement, du fait des propriétés $a + \bar{a} = 1$, ces regroupements sont égaux à la combinaison des variables qui ne change pas dans le regroupement

Exemples :

s				
$ab\backslash cd$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	1	1

$s = \bar{a} + \bar{b}c$

s				
$ab\backslash cd$	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	1	0	0
10	0	1	0	0

$s = \bar{a}\bar{c} + \bar{c}d$

s				
$ab\backslash cd$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$s = \bar{b}\bar{d}$

Traitons l'exemple vu au paragraphe précédent :

$$s = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.\bar{c}.d + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.\bar{b}.c.d + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.\bar{c}.d$$

s				
$ab\backslash cd$	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	0	0	0
10	0	0	0	0

Traitons ce tableau avec les regroupements possibles :

s				
$ab\backslash cd$	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	0	0	0
10	0	0	0	0

$s = s_1 + s_2$

$s_1 = \bar{a}.\bar{b}$

$s_2 = \bar{a}.\bar{c}$

$s = \bar{a}.(\bar{b} + \bar{c})$

Remarques :

- on voit ici que l'on a pris deux fois de suite les termes rouges que nous avions rajoutés à la main lors de la simplification par le calcul de l'expression proposée
- vu que ces tableaux ne sont pas au programme, il ne faudra pas les réaliser, mais rien ne l'interdit au brouillon ;)

A.IV. Les différents opérateurs logiques

Nous avons vu qu'il est nécessaire de savoir représenter les différentes fonctions existences afin de créer les logigrammes d'un système ainsi que leur schéma de câblage. Nous allons donc voir ici les différents opérateurs avec leurs tables de vérité, leurs logigrammes et leurs schémas de câblage.

A.IV.1 Fonctions usuelles

Soient les variables associées à la lettre e des variables d'entrée et s la sortie.

A.IV.1.a OUI

$$s = e$$

Table de vérité

e	s
0	0
1	1

Logigramme

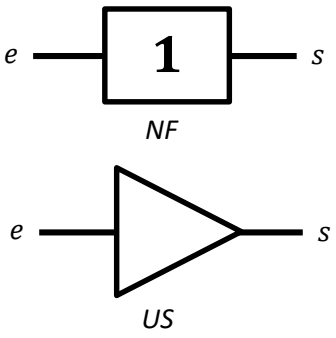


Schéma à contacts

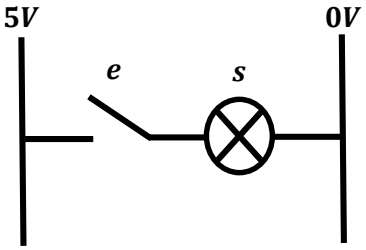
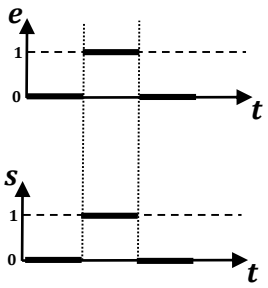


Diagramme temporel



A.IV.1.b NON

$$s = \bar{e}$$

Table de vérité

e	s
0	1
1	0

Logigramme

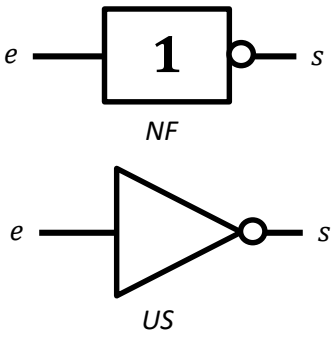


Schéma à contacts

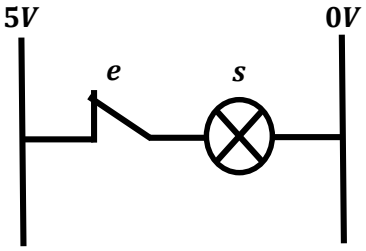
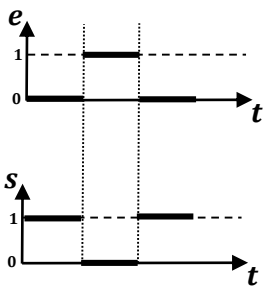


Diagramme temporel



A.IV.1.c ET

$$s = e_1 . e_2$$

Table de vérité

e_1	e_2	s
0	0	0
0	1	0
1	0	0
1	1	1

Logigramme

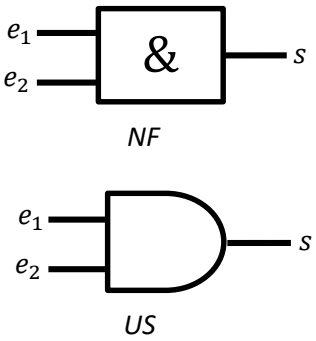


Schéma à contacts

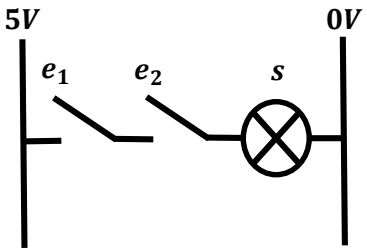
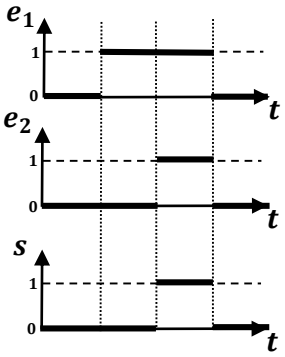


Diagramme temporel



A.IV.1.d OU

$$s = e_1 + e_2$$

Table de vérité

e_1	e_2	s
0	0	0
0	1	1
1	0	1
1	1	1

Logigramme

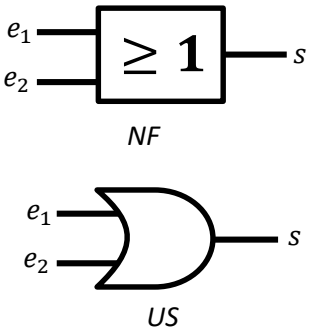


Schéma à contacts

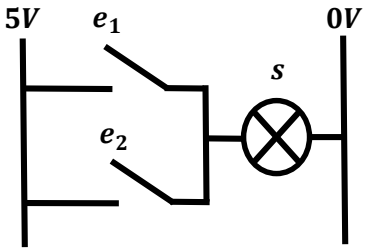
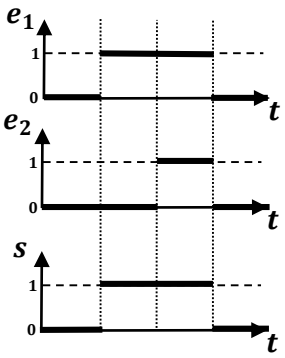


Diagramme temporel



A.IV.1.e OU EXCLUSIF

$$s = e_1 . \overline{e_2} + \overline{e_1} . e_2 = e_1 \oplus e_2$$

Table de vérité

e_1	e_2	s
0	0	0
0	1	1
1	0	1
1	1	0

Logigramme

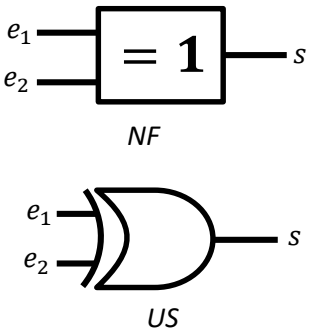


Schéma à contacts

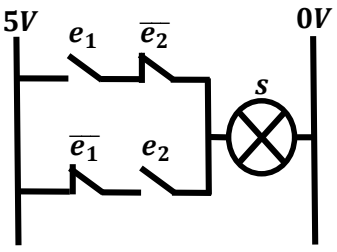
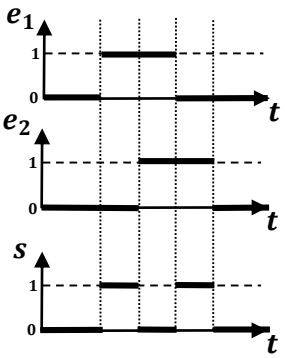


Diagramme temporel



A.IV.1.f IDENTITE

$$s = \overline{e_1}.\overline{e_2} + e_1.e_2 = \overline{e_1 \oplus e_2}$$

Table de vérité

e_1	e_2	s
0	0	1
0	1	0
1	0	0
1	1	1

Logigramme

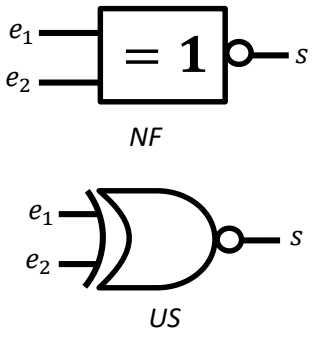


Schéma à contacts

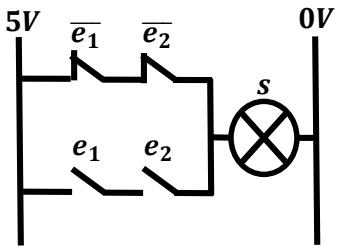
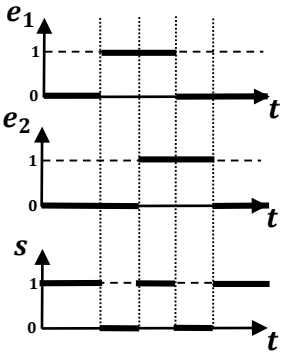


Diagramme temporel



A.IV.2 Fonctions universelles

A.IV.2.a Opérateurs NAND et NOR

On introduit deux nouveaux opérateurs NAND (NON ET) et NOR (NON OU).

A.IV.2.a.i NAND – NON ET

$$s = \overline{e_1.e_2} = \overline{e_1} + \overline{e_2}$$

Table de vérité

e_1	e_2	s
0	0	1
0	1	1
1	0	1
1	1	0

Logigramme

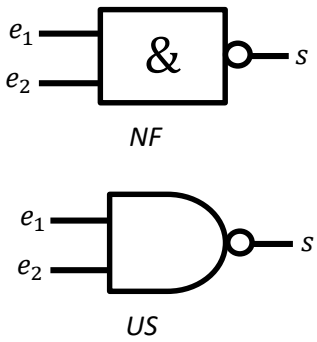


Schéma à contacts

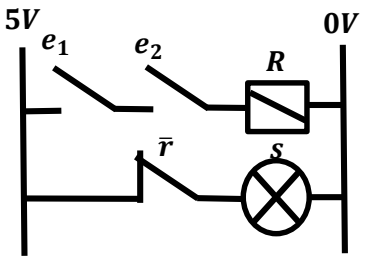
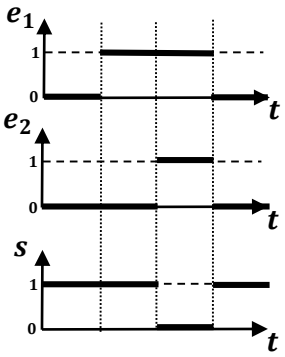


Diagramme temporel



Remarque : R est un relais associé à la variable r

A.IV.2.a.ii NOR – NON OU

$$s = \overline{e_1 + e_2} = \overline{e_1}.\overline{e_2}$$

Table de vérité

e_1	e_2	s
0	0	1
0	1	0
1	0	0
1	1	0

Logigramme

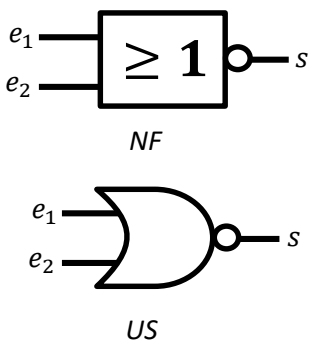


Schéma à contacts

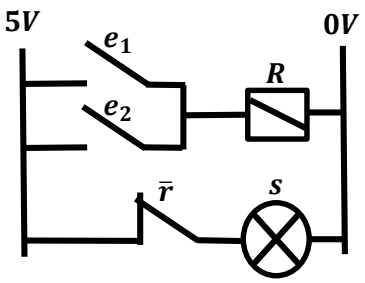
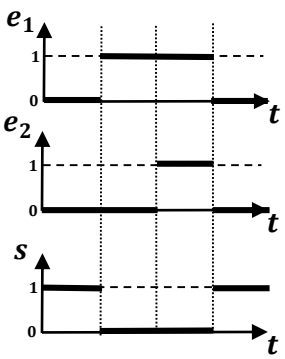


Diagramme temporel



A.IV.2.b Modèles de fonctionnement logiques à base d'un seul opérateur

A.IV.2.b.i Théorèmes de De Morgan

Le complément d'une somme logique de plusieurs variables est égal au produit du complément de chacune des variables : $\overline{\sum e_i} = \prod \overline{e_i}$

Le complément d'un produit logique de plusieurs variables est égal à la somme du complément de chacune des variables : $\overline{\prod e_i} = \sum \overline{e_i}$

A.IV.2.b.ii Utilisation

Les théorèmes de De Morgan permettent de réaliser tout modèle de fonctionnement logique d'un système à l'aide des mêmes opérateurs, que ce soit l'opérateur NAND ou NOR. Cela permet de réduire le nombre de composants différents sur un circuit électronique. Ces opérateurs sont dit **complets**.

Pour obtenir les fonctions logiques avec ces deux fonctions, le principe de base est de partir de l'égalité suivante puis de l'exploiter :

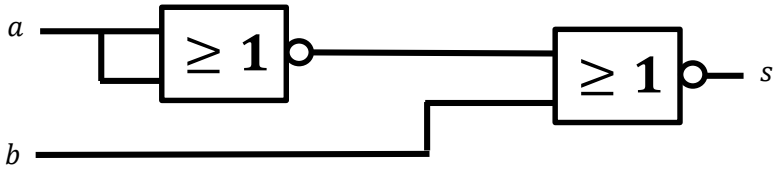
$$a = \bar{\bar{a}}$$

Pour obtenir $\bar{a}$, on doit penser à effectuer l'une des transformations suivantes :

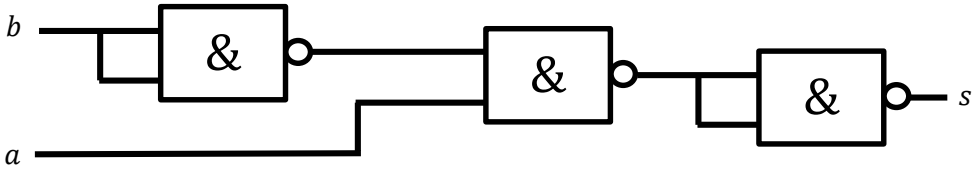
$$\bar{a} = \overline{a.a} = \overline{a + a}$$

Exemple : $s = a.\bar{b}$

Réalisation avec NOR: $s = a.\bar{b} = \overline{\overline{a.\bar{b}}} = \overline{\bar{a} + \bar{\bar{b}}} = \overline{\bar{a} + b} = \overline{\overline{a + a} + b}$



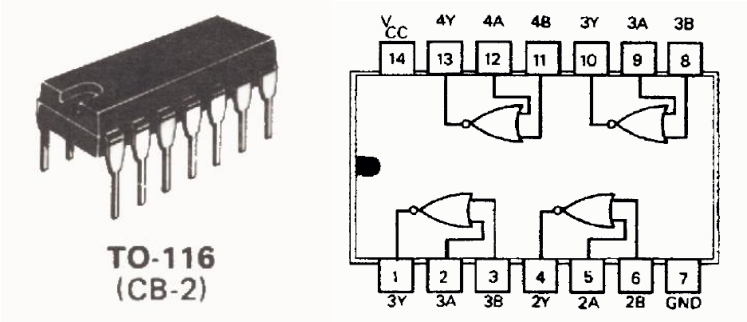
Réalisation avec NAND : $s = a.\bar{b} = \overline{\overline{a.\bar{b}}} = \overline{\overline{a.b.b}} = \overline{\overline{a.b.b}.\overline{a.b.b}}$



Remarque : pour vérifier les calculs, il faut regarder si sous chaque barre, il y a bien 2 et seulement deux termes et un symbôle . ou +

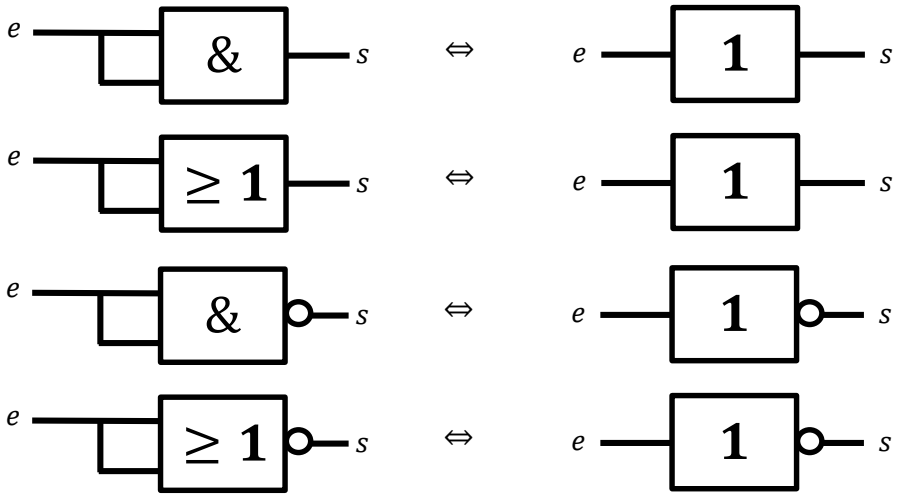
A.IV.2.b.iii Composant électrique NOR à 2 entrées

Voici un exemple de composant ajouté à un circuit imprimé qui réalise la fonction NOR 4 fois :



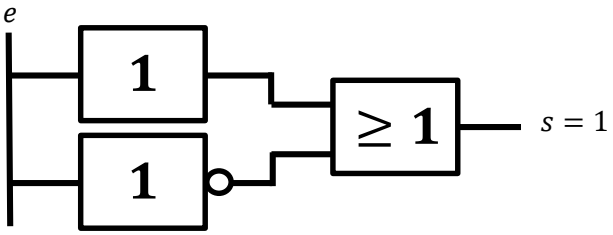
A.IV.3 Remarques sur les opérateurs

A.IV.3.a Réalisation des fonctions OUI et NON

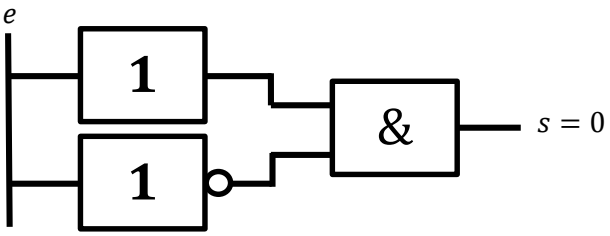


A.IV.3.b Réalisation des fonctions 1 ou 0

On peut réaliser la fonction 1 à partir d'une variable d'entrée ainsi :



On peut réaliser la fonction 0 à partir d'une variable d'entrée ainsi :



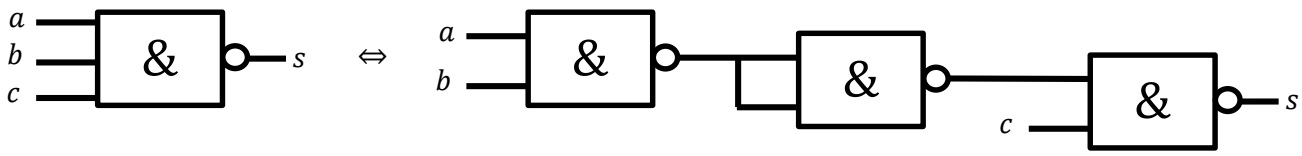
A.IV.3.c Utilisation de fonctions à 2 ou plusieurs entrées

Les opérateurs ET, OU, NON ET, NON OU sont des opérateurs qui classiquement ne possèdent que deux entrées. Il arrive que ces opérateurs soient disponibles avec un nombre d'entrées plus important. Toutefois, il faut savoir n'utiliser que des opérateurs à 2 entrées si ce sont les seuls disponibles. Les quelques exemples ci-dessous vous permettront de les utiliser.

Traitons les exemples à 3 entrées, le principe reste le même lors d'un nombre de variables plus important.

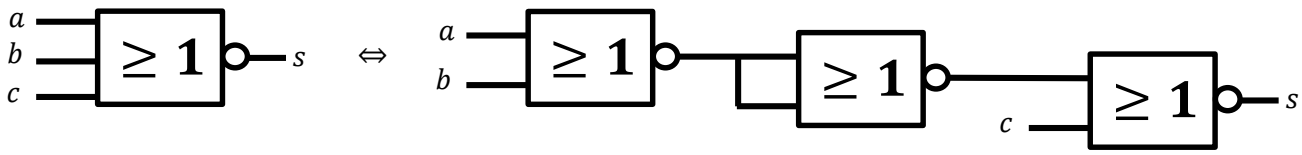
A.IV.3.c.i Equivalence NON ET à 2 et 3 entrées

$$\overline{a.b.c} = \overline{\overline{a.b}.c} = \overline{\overline{a.b}.\overline{a.b}.c}$$



A.IV.3.c.ii Equivalence NON OU à 2 et 3 entrées

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



A.V. Codage de l'information

A.V.1 Système de numération

Le système de numération le plus répandu aujourd'hui est le système **décimal**. Toutefois, dans le monde de l'automatique, on gère des informations électriques (courant présent ou non), de pression ou autre, qui sont généralement des variables TOUT OU RIEN, auxquelles on associe les variables 0 ou 1. C'est donc le système **binaire** qui sera utilisé. Il est donc nécessaire de savoir transcrire un code décimal en binaire et inversement.

A.V.1.a Digits et bases

Un nombre N est représenté par un ensemble de caractères à l'aide de **digits** dans une base b. On écrit

$$N_b$$

Les principaux systèmes de numération sont les suivants :

Nom du système	Nb de digits	Digits utilisés	Exemple : 1000
Binaire	2	0 1	$1111101000_{(2)}$
Ternaire	3	0 1 2	$1101001_{(3)}$
Quintale	5	0 1 2 3 4	$13000_{(5)}$
Octal	8	0 1 2 3 4 5 6 7	$1750_{(8)}$
Décimal	10	0 1 2 3 4 5 6 7 8 9	$1000_{(10)}$
Hexadécimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F	$3E8_{(16)}$

Dans le système binaire, on appelle

- Bit un chiffre binaire valant 0 ou 1
- Mot une suite de Bits
- Octet un mot de 8 Bits

A.V.2 Changements de bases

A.V.2.a Expression d'un même nombre dans deux bases différentes

Soit un nombre N_B constitué de p digits dans la base B et s'écrivant $N'_{B'}$ constitué de p' digits dans la base B' :

$$N_B = n_{p-1}n_{p-2}\dots n_1 n_{0_B} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \dots + n_1B^1 + n_0B^0$$

$$N_{B'} = n'_{p'-1}n'_{p'-2}\dots n'_1 n'_{0_{B'}} = n'_{p'-1}B'^{p'-1} + n'_{p'-2}B'^{p'-2} + \dots + n'_1B'^1 + n'_0B'^0$$

La conversion consiste à trouver les p' digits de $N_{B'}$ dans la base B'.

A.V.2.b Applications

Nous allons voir comment passer d'une base quelconque à la base 10 et de la base 10 à une base quelconque.

Ces passages sont simplifiés par le fait que :

- D'une base quelconque à la base 10, nous savons associer :
  - chaque digit à un nombre dans la base 10
  - chaque base B au nombre B puisqu'il est exprimé dans « notre base usuelle » 10
- De la base 10 à une base quelconque, il suffit d'appliquer la division euclidienne classique que nous connaissons dans la base 10

Pour un passage entre deux bases, on pourra simplement passer par la base intermédiaire 10.

A.V.2.b.i Base B quelconque ⟶ Base 10

Binaire ⟶ Décimal : $1111101000_{(2)}$

$$1111101000_{(2)} = 1.2^9 + 1.2^8 + 1.2^7 + 1.2^6 + 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 0.2^0$$

$$1111101000_{(2)} = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 = 512 + 256 + 128 + 64 + 32 + 8$$

$$1111101000_{(2)} = 1.10^3 + 0.10^2 + 0.10^1 + 0.10^0$$

$$1111101000_{(2)} = 1000_{(10)}$$

Hexadécimal ⟶ Décimal : $3E8_{(16)}$

$$3E8_{(16)} = 3.16^2 + 14.16^1 + 8.16^0 = 3 * 256 + 14 * 16 + 8 = 768 + 224 + 8$$

$$3E8_{(16)} = 1.10^3 + 0.10^2 + 0.10^1 + 0.10^0$$

$$3E8_{(16)} = 1000_{(10)}$$

A.V.2.b.ii Base 10 ⟶ Base B quelconque

• **Principe**

La transformation dans ce sens n'est pas aussi simple. Dans ce cas, voici la méthode à adopter :

Soit $N_{(10)}$, changer de base revient à déterminer les n_i tels que :

$$N_{(10)} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \cdots + n_2B^2 + n_1B^1 + n_0B^0$$

$$N_{(10)} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \cdots + n_2B^2 + n_1B^1 + n_0$$

Sachant que les n_i ne sont pas divisibles par B, par définition.

On peut encore écrire ce nombre sous la forme :

$$N_{(10)} = B\left(n_{p-1}B^{p-2} + n_{p-2}B^{p-1} + \cdots + n_2B^1 + n_1\right) + n_0$$

$$N_{(10)} = BQ_0 + r_0$$

On voit que n_0, qui n'est pas divisible par B, est le reste dans la division euclidienne de $N_{(10)}$ par B.

On peut continuer ainsi :

$$N_{(10)} = B\left(B\left(n_{p-1}B^{p-3} + n_{p-2}B^{p-2} + \cdots + n_2\right) + n_1\right) + n_0$$

$$B\left(n_{p-1}B^{p-3} + n_{p-2}B^{p-2} + \cdots + n_2\right) + n_1 = BQ_1 + r_1$$

On voit que n_1, qui n'est pas divisible par B, est le reste dans la division euclidienne de Q_0 par B.

Il faut donc effectuer des divisions euclidiennes successives par B, base d'arrivée, jusqu'à ce que le quotient soit nul. Les restes, du premier au dernier, correspondent aux digits en ordre inverse.

• **Exemples**

Décimal ⟶ Binaire : $1000_{(10)}$

	1000	2									
$n_0 =$	0	500	2								
	$n_1 =$	0	250	2							
		$n_2 =$	0	125	2						
			$n_3 =$	1	62	2					
				$n_4 =$	0	31	2				
					$n_5 =$	1	15	2			
						$n_6 =$	1	7	2		
							$n_7 =$	1	3	2	
								$n_8 =$	1	1	2
									$n_9 =$	1	0

$$1000_{(10)} = 1111101000_{(2)}$$

Décimal ⟶ Octal : $1000_{(10)}$

	1000	8			
$n_0 =$	0	125	8		
	$n_1 =$	5	15	8	
		$n_2 =$	7	1	8
			$n_3 =$	1	0

$$1000_{(10)} = 1750_{(8)}$$

Décimal ⟶ Hexadécimal : $1000_{(10)}$

	1000	16		
$n_0 =$	8	62	16	
	$n_1 =$	$14 = E$	3	16
		$n_2 =$	3	0

$$1000_{(10)} = 3E8_{(16)}$$

A.V.3 Codes binaires utilisant 0 et 1

Il existe une infinité de codes utilisant 0 et 1 possibles correspondant à des organisations différentes des digits d'un même nombre. Les trois codes principaux qu'il faut connaître sont les codes :

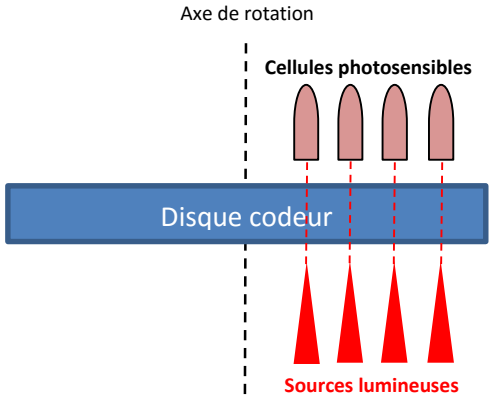
- Binaire naturel
- Binaire réfléchi ou Code Gray utilisé principalement pour coder des positions
- BCD (Binary Code Decimal) utilisé pour les calculatrices de poche

A.V.3.a Code Gray ou binaire réfléchi

En binaire naturel, un ou plusieurs digits changent en même temps entre 2 nombres successifs :

Nombre décimal	Digits code Binaire			
	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

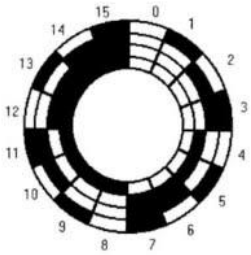
Prenons l'exemple du codage d'une position angulaire sur 16 pas par tour à l'aide d'un capteur incrémental composé d'un disque opaque composé de 4 « pistes » comportant des trous. On dispose face à chacune des pistes une source lumineuse et une cellule photosensible en face de l'autre côté du disque :



Le disque codeur est donc divisé en 4 pistes de 4 diamètres différents, puis on crée des trous lorsque l'on veut que l'information 1 transite, et on laisse la matière lorsque l'information ne doit pas passer. On peut par exemple prendre un disque en matériau transparent et coller un film opaque sur lequel on va graver (cases noires) les lieux où la lumière doit passer.

La première idée consisterait à créer des pistes selon le code binaire naturel. Ainsi, selon la position de 0 à 15, on crée des trous (cases noires) dans chaque piste lorsque le digit vaut 1 :

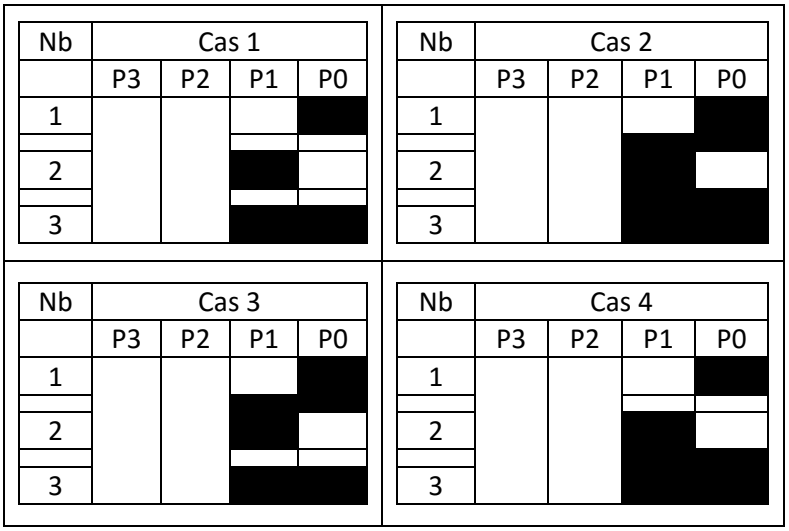
Nombre décimal	Pistes code Binaire			
	Piste 3	Piste 2	Piste 1	Piste 0
0				
1				■
2			■	
3			■	■
4		■		
5		■		■
6		■	■	
7		■	■	■
8	■			
9	■			■
10	■		■	
11	■		■	■
12	■	■		
13	■	■		■
14	■	■	■	
15	■	■	■	■



Ces pistes sont évidemment réparties circulairement sur le disque.

L'inconvénient de ce disque codeur est qu'il peut générer des erreurs du fait du changement de plusieurs bits entre deux positions successives.

En effet, imaginons deux légers décalages issus d'une fabrication imprécise des pistes autour de la position 2 et supposons que ces décalages induisent la présence des pistes suivantes (4 cas):



Pour ces 4 possibilités, l'enchaînement des positions mesurées va être le suivant :

Cas 1	Cas 2	Cas 3	Cas 4
1 – 0 – 2 – 0 – 3	1 – 3 – 2 – 3	1 – 3 – 2 – 0 – 3	1 – 0 – 2 – 3
Au lieu de 1 – 2 – 3			

Pour éviter cet inconvénient, on choisit de créer un nouveau code dans lequel un seul digit évolue à chaque changement de nombre, nommé **Code Gray**.

Nombre décimal	Digits code Gray			
	g_3	g_2	g_1	g_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

On crée les 2 premières lignes avec 0 puis 1 sur la colonne b_0
Pour les 2^1 lignes suivantes :
- on effectue une symétrie des 2^1 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

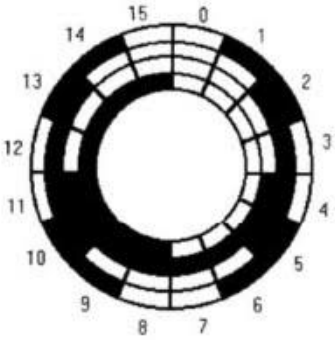
Pour les 2^2 lignes suivantes :
- on effectue une symétrie des 2^2 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

Pour les 2^3 lignes suivantes :
- on effectue une symétrie des 2^3 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

…

On obtient ainsi les pistes suivantes :

Nombre décimal	Pistes code Gray			
	g_3	g_2	g_1	g_0
0				
1				■
2			■	
3			■	
4		■	■	
5		■	■	■
6		■		■
7		■		
8	■	■		
9	■	■		■
10	■	■	■	
11	■	■	■	
12	■		■	
13	■		■	■
14	■			■
15	■			



Le gros avantage de ce code est que même s'il y a des défauts de fabrication ne dépassant pas la taille du codage d'un nombre, il n'y a aucuns risques d'erreurs de lecture de la position :

- Si deux positions différentes se chevauchent quelque peu, on ne pourra pas passer par une position intermédiaire fausse
- Pour éviter qu'il existe une zone blanche entre 2 plages, il suffira d'agrandir légèrement (valeur des défauts possibles) les deux plages afin de provoquer un chevauchement volontaire initial

Dans ces 2 cas, mais comme avec le codage en binaire naturel (à la différence d'éviter de fausses positions), les défauts induiront une légère erreur de connaissance sur la position exacte du disque codeur.

Il faudra toutefois avoir un transcodeur Gray – Binaire afin de traduire les signaux reçu codés en code Gray en binaire naturel. La table de transcription est donc la suivante, pour les 16 premiers nombres :

Nombre décimal	Digits code Binaire				Digits code Gray			
	b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
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
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
Ainsi de suite								

A.V.3.b Code BCD

Le code BCD (Binary Code Decimal) pour décimal codé binaire, est un code qui permet de s'affranchir de la transposition Binaire – Décimal qui s'avère assez lourde dès lors que les nombres sont grands (grand nombre de Bits en binaire).

Ainsi, au lieu de coder le nombre 1000 ainsi :

$$1000_{(10)} = 1111101000_{(2)}$$

Un nombre exprimé dans la base 10 est composé de digits compris entre 0 et 9. On va coder chacun de ces digits sur 4 bits (il faut coder 10 possibilités, 3 bits n'en offrent que 8, 4 en offrent 16).

Ainsi, pour 1000, on va coder 1 puis 0 puis 0 puis 0 sur 4 bits :

1				0				0				0			
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

$$1000_{(10)} = 0001000000000000_{(BCD)}$$

La lecture d'un nombre dans le code BCD s'avère bien plus simple pour nous :

$$1000100101010001_{(BCD)}$$

1	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1
8				9				5				1			

$$1000100101010001_{(BCD)} = 8951_{10}$$

Un décodeur devra donc lire les Bits 4 par 4 et transcrire chacun des chiffres entre 0 et 9 puis les assembler.

La table de transcription est donc la suivante :

Digits Base 10	Digits code BCD			
d_0	d'_3	d'_2	d'_1	d'_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1