École Supérieure
d'Électronique Appliquée
HEFF – HELB
Bruxelles

ELECTRONIQUE NUMERIQUE

Partie 1 : Les systèmes combinatoires

(UE Digital I)

Année académique 2023 – 2024

Titulaire :

Joël Pochet

Département technique

Section Electronique – Finalité Electronique Appliquée

Q1 bloc1

Table des matières

1 Introduction

- On peut diviser les circuits électroniques en 2 grandes catégories :
  analogiques et numériques
  - Les entrées et les sorties des circuits analogiques
    sont des quantités (en général des tensions) à valeurs continues,
  - Les entrées et les sorties des circuits numériques ("digitaux")
    sont des quantités à valeurs discrètes (c.-à-d. distinctes).

1.1 Représentations analogique et numérique d'une grandeur physique

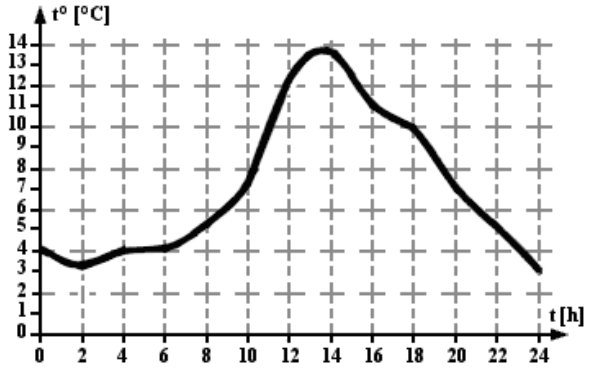


Figure 1 : La t° est une grandeur analogique

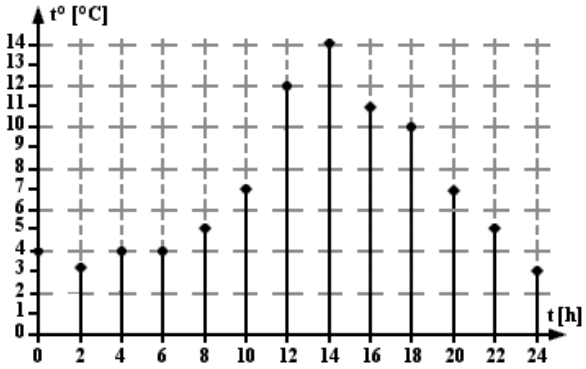


Figure 2 : Numérisation de la t°

- On peut représenter la valeur d'une grandeur physique (par ex. la température) de **2 façons** :
  - Représentation *analogique* :
    par un nombre réel = variant d'une façon continue.
    Exemple **Figure 1** :
    la t° peut prendre n'importe quelle valeur: sa variation est **continue**.
  - Représentation *numérique* = digitale [1] :
    par un nombre (généralement entier) pouvant prendre seulement des valeurs "discrètes" = variant de façon discontinue = par sauts.
    Exemple **Figure 2** : un thermomètre à affichage numérique représente la t°
    par un nombre variant par exemple par pas de 1 °C : le plus petit saut est 1 °C.
- Un système analogique peut sembler plus précis, puisqu'il peut représenter un nombre infini de valeurs. Mais, sauf à la t° absolue non atteignable de 0 K, un bruit de fond aléatoire s'ajoute de façon inévitable à tout signal : ce bruit limite la résolution de tout signal.
- Et un système numérique peut être tout aussi précis qu'un système analogique :
  même si cela en augmente le coût, on peut diminuer le pas $\Rightarrow$ augmenter [2] la résolution
  (avec la même limitation : augmenter la résolution à une valeur inférieure au bruit n'apporte plus d'information).

  Exemple : un thermomètre numérique pourrait utiliser un pas de 0,001°C :
  l'unité serait le millième de degré.

---

[1] Du mot anglais "digit" : chiffre

[2] Ici augmenter la résolution signifie améliorer la résolution. Par exemple, en passant de 0,01 °C à 0,001 °C, on améliore la résolution.

1.2 Les systèmes binaires

- Un cas particulier (extrême) de grandeur numérique est la grandeur **binaire** :
  Une variable binaire = une variable logique = un "**bit**" d'information [(1)]
  peut prendre seulement 2 valeurs (= 2 états), souvent notées
  "FALSE" et "TRUE", "OFF" et "ON", "LOW" (BAS) et "HIGH" (HAUT), ou "0" et "1".

1.3 Matérialisation physique d'un système binaire

- Une *variable binaire* est un concept mathématique (abstrait).
  Matérialiser une *variable binaire* consiste à construire un *système physique* (concret)
  dans lequel une certaine grandeur physique peut prendre *de façon stable*
  seulement 2 états bien distincts.

- Cette grandeur physique peut être mécanique, pneumatique, optique, électrique,...
  - En logique pneumatique, la grandeur est la pression de l'air dans un tuyau
    $\Rightarrow$ on s'arrange pour que la pression soit
    - ou bien très basse,
    - ou bien très élevée.
  - En électronique numérique, la grandeur est la *tension* (en anglais : *voltage*).
    La tension est certes une grandeur analogique, mais les circuits sont conçus
    afin que leur tension de sortie se stabilise rapidement, et seulement :
    - soit au niveau BAS (LOW) = une valeur proche de 0 V
    - soit au niveau HAUT (HIGH) = une valeur proche de la tension d'alimentation.

1.4 Avantage d'un système physique binaire

- Se limiter volontairement à *seulement 2* valeurs analogiques *très différentes*
  (donc faciles à créer, et à distinguer lorsqu'on mesure) offre un avantage énorme:
  le système a une *grande immunité au bruit* :
  même en présence de parasites, le risque est faible de mal interpréter la valeur d'un niveau (c.-à-d. de prendre un 0 pour un 1 , ou inversement).
- Une grande immunité au bruit augmente énormément la fiabilité : il devient possible de
  - garantir la fidélité *parfaite* (!) d'une copie
  - stocker cette copie de façon fiable, pendant longtemps
  - construire des systèmes qu'on peut étendre (par répétition et hiérarchisation)
    jusqu'à de très grandes tailles

  Osons le dire : tout ceci est impossible en analogique.

1.5 Avantages de l'électronique (numérique et analogique)

- Les technologies actuelles de fabrication permettent de miniaturiser, produire en masse,
  et diminuer les coûts de fabrication. Des progrès en termes d'écologie sont en cours.

1.6 Brève histoire de l'électronique numérique

- Voir **29 Annexe**

---

(1) De l'anglais "binary digit", chiffre binaire

1.7 L'électronique numérique moderne. Contenu de ce cours.

- Les transistors permettent de fabriquer des **portes logiques** qui réalisent électriquement les opérations fondamentales de la logique binaire (booléenne) : AND, OR, NOT et de l'arithmétique : l'addition et la soustraction

- Assembler des portes AND, OR, NOT nous conduit à la **logique combinatoire**.
  Elle permet la réalisation de codeurs, décodeurs, multiplexeurs, démultiplexeurs, …
  et par répétition de circuits simples, elle permet de créer des circuits "itératifs"
  comme les circuits de calcul arithmétique (additionneurs, soustracteurs, multiplieurs…)

- En ajoutant des rétroactions, nous entrons dans la **logique séquentielle**.
  Elle permet la réalisation de registres (mémoires), de compteurs, et d'automates variés.

- Combiner des circuits combinatoires et séquentiels nous permet de réaliser des **réseaux logiques** (GAL, PLD) **programmables par les utilisateurs**, et des **microprocesseurs**.

⇒ Tout au long de l'année, nous étudierons ces différents thèmes (avec exercices et laboratoires).

Le dernier exercice de l'année (qui occupera plusieurs séances d'exercices au cours de théorie, et plusieurs séances de laboratoire) consiste en la conception du cœur d'un microordinateur.

⇒ En 2ème vous apprendrez le langage VHDL[1] qui permet de créer des circuits hiérarchisés et itératifs de grande taille et très complexes, circuits qu'on matérialise par exemple dans des FPGA[2].

---

[1] Very high speed integrated circuit Hardware Description Language :
Langage de description du matériel pour les circuits intégrés très rapides.

[2] Field Programmable Gate Array :
Tableau de portes logiques, réseau logique (de grande taille) programmable par l'utilisateur.

1.8 Compétences à atteindre. Comment réussir le cours théorique? Auto-évaluation.

- **L'examen théorique contient seulement des exercices** ! **Ils sont directement inspirés** des **exercices donnés dans le présent syllabus, pour chaque chapitre :**
    - Faites et refaites ces exercices:
        - Comme en sport, un seul long entraînement avec beaucoup d'exercices est pénible et … inutile.
        - **Faites plutôt régulièrement 1 ou 2 exercices, le plus vite possible** Ceci est particulièrement vrai pour s'entraîner aux nombres hexadécimaux.
        - Le livre "Systèmes numériques" de FLOYD (voir bibliographie) est disponible en prêt à la bibliothèque et contient aussi de nombreux exercices, certains corrigés.
    - Vérifiez et discutez vos résultats avec vos condisciples, et vos professeurs
        - **Faites-vous expliquer** la matière par un condisciple
        - **Expliquez la matière à un condisciple; inventez un exercice pour lui !** Tous les professeurs vous le diront : donner cours, se forcer à expliquer le plus clairement possible … fait qu'on comprend mieux soi-même !
- **Vous auto-évaluer** est facile: si vous comprenez et savez résoudre les exercices du présent syllabus, si vous savez inventer des exercices semblables et les résoudre, vous réussirez.

1.9 L'électronique analogique devient de plus en plus importante !

- L'expansion des technologies numériques, des microprocesseurs et de l'informatique a été spectaculaire dans les 30 dernières années, et le sera encore dans un futur proche.

- Mais alors, l'électronique analogique est-elle dépassée, obsolète?
  Non, au contraire! Mais l'électronique analogique se spécialise :
  - Dans la conception de circuits numériques de base de plus en plus *rapides*, et de plus en plus économes en énergie.
  - Dans les interfaces entre les ordinateurs et le monde physique : prise de signaux, amplification de ces signaux avant mesure.
  - Dans les circuits de puissance (gestion de l'énergie): collecte d'énergies renouvelables, alimentation des ordinateurs, contrôle de moteurs, éclairage : leur efficacité pourrait influencer le climat futur.

1.10 Apprendre à fabriquer des produits fiables, ou faire faillite

- Un ordinateur grand public coûtant moins de 500 € peut par exemple exécuter environ 1 milliard (10^9) d'opérations par seconde : mille millions d'opérations en 1 seconde !

  Or, du point de vue du fabricant de l'ordinateur, celui-ci ne fonctionne *de façon économique* que si *tous ses circuits* – en réalité analogiques – se stabilisent en des points de fonctionnement particuliers,
  - et ceci pour des millions (10^6) de circuits,
  - pour des centaines de milliers (10^5) d'ordinateurs,
  - et chacun pendant quelques milliers (10^3) d'heures … :

  $10^9 \times 10^6 \times 10^5 \times 10^3 = 10^{23}$ … un nombre énorme !
  $\Rightarrow$ c'est seulement lorsque les circuits analogiques fonctionnent de façon *vraiment* fiable qu'on peut oublier leur fonctionnement, pour se consacrer aux problèmes numériques.

- De même, c'est seulement lorsque les circuits numériques fonctionnent de façon fiable, qu'on peut oublier leur fonctionnement pour se consacrer à l'informatique !

- Il restera alors à écrire des programmes informatiques fiables …

1.11 EXERCICES (Sujets de réflexion)

- Ce chapitre d'introduction a pour but de situer le sujet, et de vous donner quelques clés pour réussir. Comme nous n'avons vu quasi aucune matière, dans ce chapitre les exercices sont remplacés par quelques sujets de réflexion :

1. Discutez les conséquences sociales, économiques, légales d'une technologie qui permet de faire des copies parfaites

2. Discutez les conséquences sociales, économiques, légales d'une technologie qui permet de stocker des données pendant de très longues durées

3. Y a-t-il des limites pratiques au stockage de données de longue durée ?

4. Connaissez-vous des systèmes numériques de très petite taille (miniatures, microscopiques) ?

5. Connaissez-vous des systèmes numériques de très grande taille ?

6. Discutez les conséquences sociales et économiques de ces systèmes.

7. Discutez le degré de fiabilité requis pour divers produits grand public, et pour des installations industrielles.

2 Concepts numériques de base

2.1 Système binaire électronique: Association de tensions aux 2 niveaux logiques

En électronique numérique, la grandeur physique utilisée qui représente une variable binaire est généralement la tension [1] :

- En logique **positive [2]** :
  - à la valeur "0", on associe une tension V_L assez basse (LOW) : typiquement V_L = 0 V.
  - à la valeur "1", on associe une tension V_H assez haute (HIGH), typiquement $V_H = V_{CC}$ = tension d'alimentation positive, par ex. +3,3V, +5V, +12V

  avec $V_L << V_H$

- Écrire V_L = 0 V et $V_H = V_{CC}$ est un raccourci : en réalité on utilise des *plages* de tension :
  - à la valeur "0", on associe une plage de tensions basses (proche de 0V)
  - à la valeur "1", on associe une plage de tensions élevées (proches de V_{CC})

- D'une famille de composants à une autre, les valeurs de ces tensions sont différentes. Elles sont définies dans la "datasheet" du composant, publiée par son fabricant [3] .

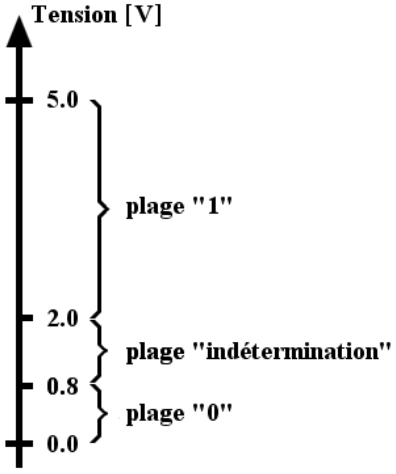


Figure 3 : Circuit TTL : plages de tension associées aux niveaux logiques 0 et 1

Cette explication est simplifiée : pour comprendre les datasheets et faire fonctionner les circuits, lisez : **Chapitre 0 Caractéristiques électriques des niveaux logiques**

---

[1] Voir cours d 'électricité et d'électronique analogique. Plus exactement, on parle ici de la tension = la différence de potentiel (aussi appelée ddp) entre le point considéré et un autre point du circuit appelé 0V ou GND (zéro volt, zéro volt logique, masse, ground).

[2] En logique *négative*, on effectue l'association inverse :
- À la valeur "0" on associe une tension élevée,
- À la valeur "1" on associe une tension basse.

Dans ce cours, on utiliser la logique positive, qui est plus répandue.

[3] On peut télécharger une datasheet depuis un site très utile comme www.alldatasheet.com , mais il est préférable de la télécharger depuis le site internet du *fabricant* du composant, afin de *toujours utiliser la dernière mise à jour* !

2.2 Timing diagram = "A.C." diagram = Chronogramme = diagramme temporel

- On illustre souvent le fonctionnement d'un système numérique avec un "diagramme temporel", ou "chronogramme", qui montre l'évolution des signaux au cours du temps.

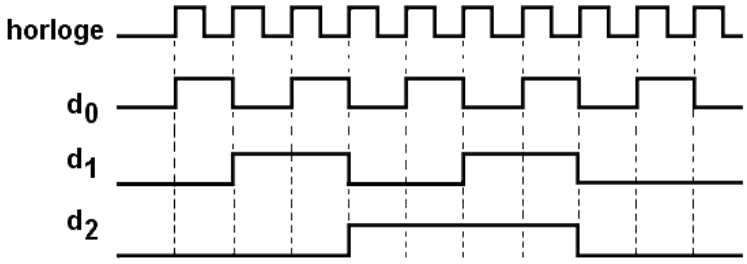


Figure 4 : Exemple de chronogramme d'un circuit numérique (compteur)
Axe horizontal : le temps

- Un chronogramme est donc ce qu'on verrait sur un oscilloscope à plusieurs traces, mais on simplifie (on idéalise) le dessin :
  - Dans un dessin simple, on ne garde que 2 niveaux de tension L et H (0 et 1)
  - Dans un dessin plus détaillé, on représente la transition d'un signal ($L \rightarrow H$, ou $H \rightarrow L$) par une ligne oblique
  - Voici quelques conventions de dessin, utilisées par exemple dans les datasheets de microprocesseurs :

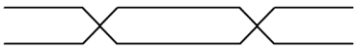
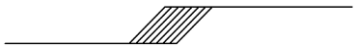
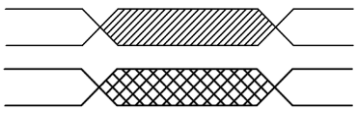
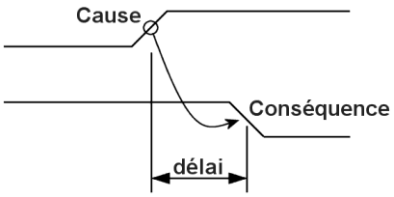
	Un signal qui monte (change de L vers H) puis qui descend (change de H vers L) .
	Un signal qui reste fixe (au niveau H ou au niveau L) .
	Un signal qui change de valeurs à 2 instants.
	Un signal initialement au niveau L, qui passe (une seule fois) à l'état H, puis y reste. Le moment précis de la transition est inconnu.
	Un signal initialement au niveau H, qui passe (une seule fois) à l'état L, puis y reste. Le moment précis de la transition est inconnu.
ou 	Un signal dont la valeur initiale est fixe (L ou H), puis est changeante/inconnue/quelconque pendant un certain temps, puis reprend une valeur fixe (L ou H) .
	Illustration d'une relation de cause à effet : le flanc d'un signal provoque le changement d'un autre signal.

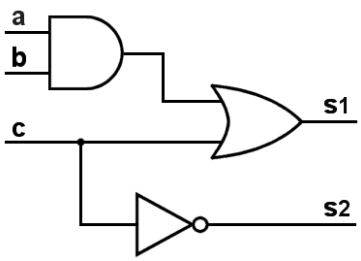
Figure 5 : Conventions de dessin utilisées dans les chronogrammes

2.3 Opérations logiques de base (en algèbre binaire)

- L'algèbre binaire [1] permet d'écrire des énoncés logiques sous forme d'équations.
  Une équation logique correspond à l'information d'égalité entre deux expressions logiques.
  Une expression logique :
  - combine des variables booléennes
  - au moyen d'opérateurs (un opérateur réalise une certaine fonction logique)
- On peut traduire une équation en *schéma* (et un schéma en équation) :
  - chaque variable binaire est représentée par la tension d'un signal électrique
  - chaque opérateur (= fonction logique) est représenté par un symbole.
- Exemple :

  s1(a,b,c) = s1 = (a AND b) OR c

  s2(c) = s2 = NOT c



Deux équations logiques
a, b, et c sont les variables (binaires)
s1 et s2 sont les fonctions [2] (binaires)
AND, OR, NOT sont les opérateurs

Schéma
a, b, et c sont les signaux d'entrée
s1 et s2 sont les signaux de sortie
Symboles des opérateurs : voir **Tableau 1** :
Opérateurs logiques de base

Figure 6 : Deux équations logiques, et le schéma correspondant à ces équations

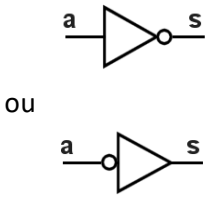
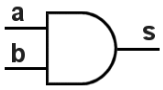
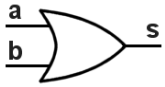
Symbole	Opérateur (Fonction)	Équation booléenne	Commentaire
	Inverter (NOT)	$s = \overline{a}$	Fonction Inverseur : Si l'entrée = 0, la sortie = 1 Si l'entrée = 1, la sortie = 0 .
	AND	$s = a \cdot b$	Fonction ET : La sortie vaut 1 si l'entrée a **et** l'entrée b valent 1
	OR	$s = a + b$	Fonction OR : La sortie vaut 1 si l'entrée a vaut 1, **ou** l'entrée b vaut 1 (ou les deux)

Tableau 1 : Opérateurs logiques de base

- Notation : a**/** ou $\overline{a}$ désigne l'inverse de a, aussi appelé : le "complément" de a
- La variable a étant binaire, on a évidemment : $\overline{\overline{a}} = a$.

[1] ou algèbre booléenne (Boole, mathématicien irlandais, http://en.wikipedia.org/wiki/George_Boole)

[2] La fonction est définie par son expression logique mais est aussi fréquemment utilisée pour identifier la variable binaire dépendante. L'utilisation du même identifiant pour la variable et le nom de la fonction apporte une facilité.

2.4 Circuits intégrés

- Des *circuits intégrés* matérialisent les opérateurs NOT, AND, OR et d'autres fonctions logiques : en les soudant sur un PCB (Printed Circuit Board, circuit imprimé), on construit physiquement le système numérique.

- Un Circuit Intégré (C.I.) est un circuit électronique encapsulé dans un seul boîtier, (et d'habitude construit sur une seule « puce » , d'habitude de silicium).

- Cette « puce » est montée à l'intérieur du boîtier qui est en plastique ou céramique. Le boîtier protège la puce des chocs, de l'humidité et de l'oxydation.

  Il existe de nombreuses variantes de boîtiers. On distingue notamment :

  - Boîtier DIL = DIP (Dual In-Line Package) : les broches du C.I. sont du type « pin through hole » : ces broches traversent le PCB
  - Boîtier SMD (Surface Mount Device) = Composant à Montage en Surface (CMS) : les broches sont courbées, elles ne traversent pas le circuit imprimé. Les boîtiers SMD modernes sont très compacts, et de plus en plus utilisés, et plus difficiles à souder car les pattes sont très rapprochées.

- Exemple : le C.I. 7408 (Quadruple AND) .

  Ce circuit existe en divers boîtiers, dont les deux ci-dessous :

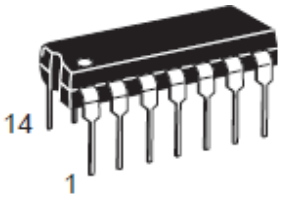
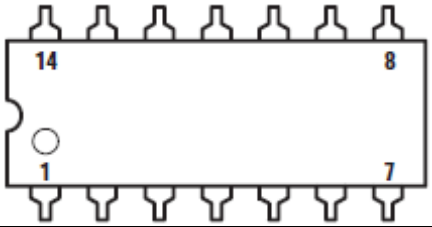
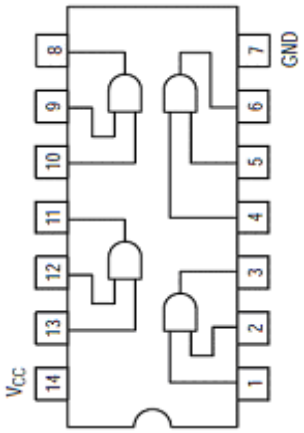
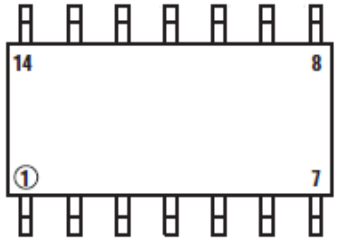
Boîtier	Numérotation des broches	Schéma interne
DIP 		
SOIC 		

Tableau 2 : La puce du 7408 est vendue dans 2 boîtiers
Le boîtier SOIC (Small Outline Integrated Circuit) est un exemple de boîtier SMD

- On fabrique des C.I. de taille et de complexité très différentes :

Scale Integration	Nombre de portes logiques sur une même puce	Exemples
SSI (Small)	jusque 10	Portes logiques de base, bascules
MSI (Medium)	jusque 100	Fonctions logiques : codeurs, décodeurs, multiplexeurs,…
LSI (Large)	jusque 1000	Mémoires,…
VLSI (Very Large)	jusque 100000	Mémoires,…
ULSI (Ultra Large)	> 100000	Mémoires, microprocesseurs, …

Tableau 3 : Classement des C.I. par complexité

Exemple : le C.I. 7408 (Quadruple AND) est un circuit SSI .

- On fabrique des C.I.
  - avec différents types de transistors (transistor bipolaire, transistor MOSFET),
  - avec différents matériaux semi-conducteurs (silicium, arséniure de gallium, …)
  - on utilise ces transistors de différentes façons (en saturation, ou non)

  Les choix de fabrication influencent le coût, la consommation, et la vitesse des C.I.
  (voir cours d'électronique analogique).

- Vous utiliserez surtout des circuits du type TTL ou CMOS :

Type	Technologie des transistors	Indications
TTL	bipolaire	SSI, MSI. Robustes. Survivent assez bien aux décharges électrostatiques
CMOS	MOSFET	Plus compacts, consomment moins d'énergie. Pour tous circuits (SSI jusqu'ULSI). Assez fragiles (notamment en cas de décharge électrostatique)

Tableau 4 : Principales technologies de C.I.

Exemple : le 7408 existe en TTL (7408, 74LS08) et en CMOS (74HC08, 74LVC08,…)

- On distingue 2 grands types de C.I. :
  - C.I. à fonction *fixe* : ses fonctions logiques sont figées, impossible à modifier.
  - C.I. à fonction *programmable*.

  Exemple : le 7408 est un circuit à fonction fixe. Les PAL, les FPGA sont programmables.

- Voir aussi "Systèmes numériques", Thomas L FLOYD, 9ème édition, pg. 151
  et "Le technicien en électronique", C. Cimelli, R. Bourgeron, pg. 273.

2.5 Matérialisation de fonctions booléennes autrement qu'en électronique

- Nous matérialiserons les circuits logiques par des C.I., mais d'autres possibilités existent:
  - Circuits électroniques à composants discrets : DTL et RTL (Diode Transistor Logic, Resistor Transistor Logic) qui utilisent des résistances et des diodes,
    et des transistors comme amplificateurs / inverseurs
    (voir **19** : **Annexe : fonctions logiques élémentaires en DTL et RTL**
    et le cours d'électronique analogique).
  - Circuits électriques à interrupteurs et relais, avec des lampes pour affichage
  - Circuits pneumatiques
  - Circuits optiques
  - …

- Exemple : circuit électrique matérialisant la fonction logique "AND à 2 entrées" :
  - La sortie s du circuit vaut 1 si l'entrée a ET l'entrée b valent 1 ; sinon s = 0 .
    On résume la description de la fonction s(a,b) en une "Table de Vérité" [(1)] :

Entrées b	Entrées a	Sortie s = a . b
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 5 : Table de vérité de la fonction AND

  - s est matérialisée par une lampe : 1 = lampe allumée, 0 = lampe éteinte
  - La variable d'entrée a est matérialisée par le switch (interrupteur) Sw_a ,
    la variable d'entrée b est matérialisée par Sw_b .
    Un switch fermé représente l'état 1 , un switch ouvert représente l'état 0 .

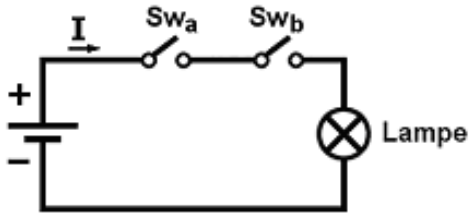


Figure 7 : Un circuit électrique qui matérialise la fonction ET
Ici, les 2 switches sont représentés ouverts (état 0)
Si [Sw_a est fermé (état 1)] ET [Sw_b est fermé (état 1)] alors [la lampe est allumée (état 1)]

(1) La sortie est une fonction des variables d'entrée.
La table de vérité d'une fonction booléenne *spécifie la valeur de la fonction*,
pour chacune des combinaisons possibles des variables

2.6 Intérêts de l'algèbre binaire

- Nous avons vu que l'algèbre binaire permet de représenter
  les relations entre les entrées et la sortie (ou les sorties) d'un circuit logique
  sous une forme mathématique : équation(s), ou table(s) de vérité.

  Ceci offre plusieurs avantages :

  - La représentation mathématique (équation(s) ou table(s))
    est indépendante de la technologie qui sera choisie pour réaliser la fonction
  - Dans le cas d'une fonction d'un nombre limité de variables[(1)] d'entrée ;
    l'être humain est capable (en exploitant quelques propriétés mathématiques)
    de simplifier la fonction au maximum, avant de la réaliser de façon concrète
  - Dans le cas d'une fonction de nombreuses variables,
    il est possible de simplifier la fonction à l'aide de programmes informatiques.

- Toutefois, l'utilisation de variables binaires qui n'ont que deux valeurs possibles
  rallonge les expressions, qui deviennent rapidement illisibles pour l'être humain.

  Aussi, avant de passer au **Chapitre 5 Systèmes combinatoires et fonctions combinatoires de base**,
  nous allons introduire des notations plus compactes que le binaire.

  C'est l'objet du **Chapitre 3 Représentation des entiers naturels: numération de position :**
  nous y verrons notamment le code hexadécimal :

  - écrire des nombres en hexadécimal est beaucoup plus court qu'en binaire,
  - la conversion binaire $\rightarrow$ hexadécimal, et la conversion hexadécimal $\rightarrow$ binaire
    sont très simples.

  et c'est aussi l'objet du **Chapitre 4 Divers codes .**

[(1)] Disons : 4 ou 5 variables

2.7 EXERCICES

8. Énoncé : Dessinez le circuit électrique (avec des interrupteurs et une lampe) réalisant la fonction "OR à 2 entrées"

9. Dessinez les circuits d'interrupteurs correspondant aux calculs logiques suivants :
   - $s = (a + b) . c$

   - $s = [(a + b) . c] + d$

   - $s = [a + (b . c)] + d$

10. Reprenez les équations de l'exercice ci-dessus et réalisez les circuits en câblant des opérateurs logiques de base.
    - s = (a + b) . c

    - s = [a + (b . c)] + d

    - s = { [a + (b . c)] + d } + e

11. Dessinez un chronogramme montrant l'entièreté des états possibles de l'entrée et de la sortie de l'opérateur logique NOT

12. Dessinez un chronogramme montrant l'entièreté des états possibles des 2 entrées et de la sortie de l'opérateur logique AND

13. Dessinez un chronogramme montrant l'entièreté des états possibles des entrées et de la sortie de l'opérateur logique OR

3 Représentation des entiers naturels: numération de position

3.1 Introduction

- Un **bit** (un chiffre binaire) ne peut avoir que 2 valeurs (différentes) : 0 et 1

  $\Rightarrow$ pour représenter un nombre plus grand que 1, on utilise *un groupe, un ensemble de bits* :
  - 1 seul bit peut prendre 2 valeurs : il peut représenter les nombres entiers 0 et 1
  - Un groupe de 2 bits peut prendre 4 valeurs, et représenter les entiers 0, 1, 2, 3
  - Un groupe de 3 bits peut prendre 8 valeurs, et représenter les entiers 0 à 7
  - etc. : **un groupe de n bits peut prendre 2^n valeurs différentes**

3.2 Rappel : Base 10 (numération décimale)

- Etant enfants, nous avons appris le système de numération "de position", "en base 10" :

  Par exemple, écrire N = 8625 signifie : $N = 8 . 10^3 + 6 . 10^2 + 2 . 10^1 + 5 . 10^0$:

Rang :	3	2	1	0
Pondération :	10^3	10^2	10^1	10^0
Chiffre :	**8**	**6**	**2**	**5**

Tableau 7 : Pondération des chiffres dans un nombre écrit en base 10

  autrement dit : chaque chiffre est *pondéré* ; le poids d'un chiffre dépend de son rang :
  - Si le chiffre figure tout à droite, son rang est 0 , son poids est 1 (10^0) .
    Dans l'exemple, le chiffre 5 a le poids 1
  - A chaque déplacement d'un rang vers la gauche, le poids est multiplié par 10 .
    Dans l'exemple, le chiffre 2 a le poids (10^1), le chiffre 6 a le poids (10^2), etc.

- Notes :
  - En base B = 10 , il existe 10 chiffres : les chiffres de 0 à 9 ;
    le chiffre le plus élevé est le chiffre 9 , et on a $9 = B - 1$.
  - Ajouter un zéro à droite du nombre multiplie le poids de chaque chiffre par 10 ,
    et donc multiplie le nombre lui-même 10 : $86250_{10} = 10 \times 8625_{10}$

3.3 Base B quelconque

- La base 10 est seulement un cas particulier :
  on peut représenter tout nombre dans n'importe quelle base B entière (B > 1)
- Dans une base B , les chiffres ont une valeur comprise entre 0 et (B-1) , et

Le nombre entier naturel N représenté par n chiffres $C_{n-1}\, C_{n-2} \ldots C_1\, C_0$

vaut : $\mathbf{N = C_{n-1}\, B^{n-1} + C_{n-2}\, B^{n-2} + \ldots + C_1\, B^1 + C_0\, B^0 = \sum_{i=n-1}^{0} C_i . B^i}$ **Équation 1**

  Résumé : chaque chiffre est pondéré, et le poids d'un chiffre dépend de son rang :
  - si le chiffre figure tout à droite, son rang est 0 , son poids est 1 ($= B^0$) .
  - à chaque déplacement d'un rang vers la gauche, le poids est multiplié par la base .

3.4 Base 2 (binaire)

- En base 2, un chiffre (appelé aussi "bit") a seulement **2 valeurs possibles :** 0 ou 1 .
  Exemple : $11101_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 4 + 1 = 29_{10}$

  Pour améliorer la lisibilité, l'usage est de regrouper les bits par tranche de 4 bits,
  en partant de la droite : on écrit par exemple : $10\ 0111\ 1101_2$

3.5 Base hex (hexadécimale)

- Pour une base $B > 10$, les chiffres C_i ayant une valeur comprise entre 0 et $B-1$,
  certains chiffres sont supérieurs à 9 $\Rightarrow$ on a besoin de nouveaux symboles ;
  l'usage est d'utiliser les premières lettres de l'alphabet :
- En base hex, B = 16, les 16 chiffres sont : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F :
  les symboles A, B, C, D, E, F sont les "chiffres" supérieurs à 9, leurs valeurs sont :
  - $A_{hex} = 10_{déc}$
  - $B_{hex} = 11_{déc}$
  - $C_{hex} = 12_{déc}$
  - $D_{hex} = 13_{déc}$
  - $E_{hex} = 14_{déc}$
  - $F_{hex} = 15_{déc}$

  Exemple : $3BD_{hex} = 3 \cdot 16^2 + 11 \cdot 16^1 + 13 \cdot 16^0 = 768 + 176 + 13 = 957_{déc}$

Déc.	Hex	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Tableau 8 : Entiers de 0 à 15 en dec, bin, hex

- Pour info : d'autres bases B > 10 sont possibles, mais jamais utilisées.
  Par ex., en base 20, on utiliserait 20 symboles : 0, 1, 2, … 7, 8, 9, A, B, C, D, E, F, G, H, I, J .

3.6 Utilisation de différentes bases

- $245_{10} \neq 245_{hex}$: lorsqu'on utilise plusieurs bases, il FAUT spécifier la base utilisée !

Pour cela, on ajoute un indice au nombre. L'indice indique la base utilisée, et on peut l'écrire en chiffres ou en lettres (en anglais, français, ...), et en abrégé.

Base	Appellation	Notation
10	Décimale	245_{10} , $245_{décimal}$, $245_{déc}$, 245_{dec} , 245_{d}
2	Binaire	$1111\,0101_{2}$, $1111\,0101_{binaire}$, $1111\,0101_{binary}$, $1111\,0101_{bin}$, $1111\,0101_{b}$
8	Octal	365_{8} , 365_{octal} , 365_{oct} (Note : on utilise plutôt l'hex, plus compact).
16	Hexadécimal (hex)	$F5_{16}$, $F5_{h}$, $F5_{hex}$ Certaines notations sont inspirées des langages de programmation : F5H, 0xF5, 0XF5

Tableau 9 : Le même nombre décimal 245, dans quelques bases usuelles

- La base 2 est utilisée par les systèmes logiques. Elle est pénible pour les êtres humains, car les nombres y sont représentés par de longues suites de bits
- Les circuits numériques modernes fonctionnent en binaire, mais dans les documentations, on écrit souvent les nombres binaires en hex :
  - la notation hex est une forme commode et condensée du binaire, et
  - nous verrons que les conversions $16 \leftrightarrow 2$ sont très faciles, immédiates

3.7 Conversion d'un entier naturel d'une base à une autre base

3.7.1 Base B (par ex. Hex ou Bin) $\rightarrow$ Décimal

- On applique l' Equation 1: $N = C_{n-1} B^{n-1} + C_{n-2} B^{n-2} + \ldots + C_1 B^1 + C_0 B^0$

Exemple : Convertir $3CE_h$ en décimal :

$$3CE_h = 3 \cdot 16^2 + 12 \cdot 16^1 + 14 \cdot 16^0$$
$$= 768_{10} + 192_{10} + 14_{10} = 974_{10}$$

3.7.2 Base $B = 2^n$ (par ex. Hex) $\leftrightarrow$ Binaire

- Passer d'une [base puissance de 2] à la base 2 , ou inversement, est rapide :

> Si $B = 2^n$, chaque chiffre de base B est équivalent à une tranche de n chiffres binaires

- **Hex $\rightarrow$ Bin** :
  - $B = 16 = 2^4 \Rightarrow$ *on remplace chaque chiffre hex par 4 chiffres binaires*.
  - Pour accélérer l'opération, *apprenez par cœur* les nombres de 0 à 15 en bin et hex :

Déc.	Hex	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Réf : Tableau 8 : Entiers de 0 à 15 en dec, bin, hex

  - Exemples :

    Convertir $F5_h$ en binaire. Réponse : $F5_h = 1111\,0101_2$

    Convertir $78DE_h$ en binaire. Réponse : $78DE_h = 0111\,1000\,1101\,1110_2$

- **Bin $\rightarrow$ Hex** :
  - Utilisez le même tableau
  - *Attention: constituez les groupes de chiffres binaires en partant du chiffre le plus à droite (unités), puis déplacez-vous progressivement vers la gauche par 4 chiffres à la fois !*
  - Exemple : convertir 11011011100010_2 en hex :
    On groupe les bits en tranches de 4 bits, en commençant par le côté droit :
    $11011011100010_2 = (00)11\ 0110\ 1110\ 0010_2 = 36E2_h$

3.7.3 Décimal → base B (B entier quelconque, et en particulier 16 ou 2)

- Algorithme de conversion d'un entier décimal ≥ 0 vers une base B entière :

  Diviser le nombre à convertir par la base d'arrivée B (division entière, notez le reste) ;
  recommencez avec le quotient (et notez le reste), cela jusqu'à obtenir un quotient nul ;
  Puis écrire la suite des restes *dans l'ordre inverse* où vous les avez obtenus.

- *Comment retenir cette procédure ? Par un exemple :*
  La procédure s'applique aussi pour retrouver la succession des chiffres d'un nombre décimal.
  Ainsi, pour trouver la suite des chiffres du nombre 743_{dec} :
  je divise 743 par la base 10 ⇒ quotient = 74, reste = 3 ⇒ c'est le dernier chiffre du nombre
  je divise 74 par 10 ⇒ quotient = 7, reste = 4
  je divise 7 par 10 ⇒ quotient = 0, reste = 7

- Pour organiser les calculs, certains auteurs dessinent un tableau des résultats successifs :
  - Colonne de gauche : on écrit le nombre de départ, puis la suite des quotients
  - Colonne de droite : on écrit la suite des restes

Exemple : convertir $7925_{déc}$ en hex : ⇒ on fait des divisions successives par 16 :

Quotients	Restes
7925	$5 = 5_{hex}$
495	$15 = F_{hex}$
30	$14 = E_{hex}$
1	$1 = 1_{hex}$
0	

(en effet : 7925/16 = 495 reste 5)

⇒ $7925_{déc} = 1EF5_{hex}$

Tableau 10 : Conversion en hexadécimal

- Exemple 2 : convertir $424_{déc}$ en binaire : ⇒ on doit faire des divisions successives par 2

Quotients	Restes
424	0
212	0
106	0
53	1
26	0
13	1
6	0
3	1
1	1
0	

Note : le reste de la division d'un nombre par 2
- vaut 0 si le nombre est pair,
- vaut 1 si le nombre est impair

⇒ $424_{10} = 1\ 1010\ 1000_2$

Tableau 11 : Conversion en binaire

3.7.4 Base hex = intermédiaire entre décimal et binaire

- Pour la conversion $2 \rightarrow 10$, le plus facile est de passer par le hex : on fait $2 \rightarrow 16 \rightarrow 10$
- Pour la conversion $10 \rightarrow 2$, on passe aussi souvent par le hex : on fait $10 \rightarrow 16 \rightarrow 2$
- Même si en réalité les circuits numériques fonctionnent en binaire,
  dans les documentations on écrit souvent les nombres binaires en hex
  = sous une forme plus condensée, plus lisible.

3.7.5 Pour information : Base B1 quelconque $\rightarrow$ autre base B2 quelconque

- En lisant de vieilles documentations, on doit parfois convertir de l'octal en hex.
  Ces deux bases étant des puissances de 2, le plus simple est de passer par le binaire :
  B1 $\rightarrow$ Binaire $\rightarrow$ B2

  Exemple : convertir 365_{oct} en hex :

  1363_{oct} = $1\ 011\ 110\ 011_2$ (remplacement de chaque chiffre oct. par une tranche 3 bits)
  = $10\ 1111\ 0011_2$ (regroupement par tranche de 4 bits, en partant de la droite)
  = $2F3_{hex}$

- Nous ne rencontrerons pas ce problème, mais
  pour passer d'une base B1 quelconque à une autre base B2 quelconque,
  le plus facile est sans doute de passer par le décimal :
  B1 $\rightarrow$ Décimal $\rightarrow$ B2

3.8 EXERCICES

14. Connaissez-vous des bases de numération (autres que 2, 10, 16) qui ont été utilisées dans le passé, ou le sont encore partiellement aujourd'hui ?

15. Convertir 40725_8 en hex

16. Convertir $7925_{déc}$ en hex puis en binaire

17. Vérifiez votre résultat de la conversion de $7925_{déc}$ en hex

18. Convertir $849_{déc}$ en hex puis en binaire

19. Vérifiez votre résultat de la conversion de $849_{déc}$ en hex

4 Divers codes

4.1 Codes. Codes binaires

- L'action de faire correspondre une *suite de symboles* à un nombre, une lettre ou un mot s'appelle : "coder". La suite de symboles s'appelle un "code".

  Exemples :

  - Le système de numération de position en base 2 exposé ci-dessus permet d'écrire, de coder un entier positif en binaire. On l'appelle *code binaire naturel* ou *code binaire pur*
  - Le code BCD est un autre code binaire : c'est une *autre* façon de représenter un entier par une suite de bits.

4.2 Le codage BCD

- Le code appelé "Décimal Codé Binaire" (en abrégé **BCD**, Binary Coded Decimal) considère chaque chiffre d'un nombre décimal séparément, et le code en binaire pur.

Chiffre décimal	Chiffre codé en BCD			
	d	c	b	a
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

Tableau 12 : Code BCD

Il existe seulement 10 chiffres décimaux : 0 à 9
Le code BCD utilise les 10 combinaisons de 4 bits qui leur correspondent.

Les 6 autres combinaisons de 4 bits qui correspondent aux nombres décimaux 10 à 15 (c.-à-d. 1010, 1011, 1100, 1101, 1110 et 1111) ne sont pas utilisées.

- Exemple : convertir 39_{10} en BCD :
  On convertit séparément chaque chiffre: $3 = 0011_2$, $9 = 1001_2$ $\Rightarrow$ $39_{10} = 0011\ 1001_{BCD}$
- Notez que $39_{10} = 10\,0111_2$: le code binaire pur d'un nombre diffère de son code BCD !
- Dans un ordinateur, on peut utiliser le code qu'on veut pour représenter des données : le programmeur doit adapter son programme au type de données qu'il veut traiter.
- La conversion Base 10 $\leftrightarrow$ BCD est facile pour l'être humain, mais le code BCD est moins compact que le binaire pur, et les opérations arithmétiques nécessitent de petits programmes $\Rightarrow$ les calculs en BCD sont lents.

4.3 Le code Gray (code binaire réfléchi) (BRGC)

- Vocabulaire : BRGC signifie = "Binary-Reflected Gray Code",
  en français: "code de Gray binaire réfléchi". Nous verrons pourquoi ci-dessous.

4.3.1 Problème de la numérotation binaire naturelle

- En binaire naturel, deux entiers successifs peuvent différer par plusieurs bits :
  Exemples : $1_{déc} = 001_{bin}$ et $2_{déc} = 010_{bin}$ ont 2 bits différents.
  Ceci crée parfois un risque de confusion :
- Imaginons une station météo qui indique la direction du vent par un angle en 3 bits :
  chaque angle vaut $360° / 8 = 45°$. Un circuit "codeur angulaire" encode les angles successifs par des nombres binaires successifs : 000_{bin} désigne le Nord, 100_{bin} le Sud, etc.
- De la position 011_{bin} à la position 100_{bin}, les 3 bits changent … or, le codeur angulaire est un système physique : ces 3 bits mettent un certain temps à changer! *Et ce temps diffère légèrement pour chacun des bits* $\Rightarrow$ *il apparaît des états transitoires difficiles à interpréter* :
  Exemple : de 100_{bin} à 011_{bin}, si le bit de poids fort change le premier, il apparaît transitoirement 000_{bin} (Nord), alors que 100_{bin} et 011_{bin} désignent des directions Sud !

4.3.2 Définition du code Gray

- Dans le code Gray, *un seul bit change entre 2 codes successifs.*
- Voici les codes Gray correspondant aux premiers entiers, et une méthode pour les trouver

$N_{déc}$	Code Gray G_{bin} $G_3G_2G_1G_0$	Action
0	0000	On commence le premier code Gray avec tous des 0
1	0001	Pour le code Gray suivant : on change le bit de poids faible : $G_0 = 1$
2	0011	Pour les 2 codes Gray suivants : on a déjà utilisé les 2 états de G_0 $\Rightarrow G_1 = 1$; on copie les bit G_0 des codes au-dessus, *dans l'ordre inverse (copie en miroir; c'est pourquoi le code de Gray est appelé : code binaire "réfléchi")*
3	0010	
4	0110	Pour les 4 codes Gray suivants : on a déjà utilisé les 4 états de G_1G_0 $\Rightarrow G_2 = 1$, et copie miroir des bits G_1G_0 au-dessus
5	0111	
6	0101	
7	0100	
8	1100	Pour les 8 codes Gray suivants : on a déjà utilisé les 8 états de $G_2G_1G_0$. $\Rightarrow G_3 = 1$, et copie miroir des bits $G_2G_1G_0$ au-dessus etc.
9	1101	
…	…	… etc.

Tableau 13 : Code Gray (colonne de droite)
Un seul bit change entre les codes Gray correspondant à 2 entiers successifs !

4.3.3 Pour information – Distance de Hamming entre 2 suites de bits de même longueur

- Définition : La "distance [(1)] entre 2 suites de bits de même longueur
  est le nombre de bits qui diffèrent entre ces 2 suites.

- Exemples:

  - Soient P et Q deux suites de 7 bits qui valent:

    P = 0 1 1 1 0 0 1
    Q = 0 1 0 1 1 1 1
    ____

    $\Rightarrow P \oplus Q$ = 0 0 1 0 1 1 0 $\Rightarrow$ distance entre P et Q : 3
    (P et Q ont 3 bits différents)

  - Soient R et S deux suites de 6 bits qui valent:

    R = 1 1 1 0 0 1
    S = 1 0 1 0 0 1
    ____

    $\Rightarrow R \oplus S$ = 0 1 0 0 0 0 $\Rightarrow$ distance entre R et S : 1
    (R et S ont 1 seul bit différent)

- Dire "Deux suites de bits diffèrent par 1 et 1 seul bit"
  est équivalant à dire : "Ces deux suites de bits ont une distance 1"

- Soit l'ensemble des nombres naturels B de n bits (il y en a 2^n).

  A chaque nombre B on fait correspondre un code G (un autre nombre de n bits).

  Définition: Un "code à distance unitaire" est tel que

  - Deux codes successifs ont une distance unitaire (ils diffèrent par 1 et 1 seul bit)
  - De même, le dernier et le premier code ont une distance unitaire
  - Tous les G sont différents les uns des autres

  Le code "Gray" est un cas particulier de code à distance unitaire, qu'on appelle aussi :
  "Binary-Reflected Gray Code" (BRGC), en français: "code de Gray binaire réfléchi",
  car on génère les codes successifs en faisant des copies successives "en miroir"
  de morceaux de la table des codes.

- La distance de Hamming intervient non seulement dans la définition du code Gray,
  mais elle est aussi très importante dans la mise au point des "codes correcteurs d'erreur"
  en télécommunication numérique

4.3.4 Ciruit convertisseur Binaire pur $\leftrightarrow$ Gray.

- Voir **Annexe 26 : c**ircuit itératif "Générateur/Décodeur de code Gray"
  (c.-à-d. convertisseur Binaire pur $\leftrightarrow$ Gray).

4.3.5 Note : Numérotation des cellules dans une table de Karnaugh

- La numérotation des cellules dans une table de Karnaugh (**Chapitre 12**)
  est liée au code Gray

[(1)] ou plus précisément : la "distance de Hamming" , en anglais: the "Hamming distance"

4.4 Les codes alphanumériques. ASCII, Unicode

- Nous avons vu plusieurs codes binaires qui correspondent à différentes façons de représenter (coder) un nombre entier positif en binaire.

  Mais un nombre (code) binaire peut aussi représenter autre chose qu'un nombre entier positif.
  Exemples :

  - Le code Morse représente une lettre par une suite de points et de traits :
    c'est un code binaire (il n'a que 2 symboles), mais il ne nous intéresse pas.
  - Dans un ordinateur, des nombres binaires (parfois de 16 bits ou plus) représentent souvent des informations numériques ou non, au moyen d'un code. Par exemple :
    - un nombre (entier arithmétique ou algébrique, ou réel),
    - l'adresse (le numéro) d'un emplacement en mémoire,
    - le code d'une instruction,
    - un code correspondant à un caractère alphabétique,
    - un groupe de bits indiquant l'état dans lequel se trouvent des signaux internes, ou des signaux d'entrées/sorties de l'ordinateur.

- Les codes qui correspondent à des lettres, des signes de ponctuation et des caractères spéciaux sont dits *alphanumériques* . L' ASCII (American Standard Code for Information Interchange) est le code alphanumérique le plus répandu.

  - Le code *ASCII standard* (**Tableau 14** page suivante) est un code sur **7 bits**
    - ce code permet donc de représenter 2^7 = 128 éléments codés.
    - Pour l'anglais, 128 codes suffisent pour désigner les chiffres,
      les lettres (majuscules et minuscules), les signes de ponctuation,
      et quelques commandes
    - L'ASCII standard n'inclut pas les caractères lettres accentuées (é, è, à, …)
  - Le code *ASCII étendu* (voir Annexe **Tableau 71**) est codé sur **8 bits**
    - on dispose de 2^8 = 256 éléments codés.
    - Cette extension du code ASCII n'est pas standard.
      Le français nécessite des lettres accentuées : é, è, à...
      L'allemand en nécessite d'autres : ü, ä...
      Il existe donc de nombreuses variantes locales du code ASCII 8 bits !

    Le code ASCII est très utilisé dans les programmes simples,
    notamment par les programmes qui affichent des caractères à largeur fixe,
    indépendants les uns des autres.

  - L' *Unicode* est un code plus compliqué, sophistiqué, et bien plus général :
    l'Unicode permet de traiter les caractères arabes, hindous, chinois, …
    voir www.unicode.org

    Sur un PC, les jeux de caractères UTF-8, -16, et -32 sont compatibles Unicode.
    L'ASCII est un (petit) sous-ensemble de l'Unicode.

4.4.1 Code ASCII en 7 bit

Hex	Char	Comment	Ctrl	Hex	Char	Hex	Char	Hex	Char
00	NULL	Null character	^@	20	space	40	@	60	‘
01	SOH	Start of Header	^A	21	!	41	A	61	a
02	STX	Start of Text	^B	22	"	42	B	62	b
03	ETX	End of Text	^C	23	#	43	C	63	c
04	EOT	End of Transmission	^D	24	$	44	D	64	d
05	ENQ	Enquiry	^E	25	%	45	E	65	e
06	ACK	Acknowledgement	^F	26	&	46	F	66	f
07	BELL	Bell	^G	27	'	47	G	67	g
08	BS	Backspace	^H	28	(	48	H	68	h
09	HT	Horizontal Tab	^I	29	)	49	I	69	i
0A	LF	Line Feed	^J	2A	*	4A	J	6A	j
0B	VT	Vertical Tab	^K	2B	+	4B	K	6B	k
0C	FF	Form Feed	^L	2C	,	4C	L	6C	l
0D	CR	Carriage Return	^M	2D	-	4D	M	6D	m
0E	SO	Shift Out	^N	2E	.	4E	N	6E	n
0F	SI	Shift In	^O	2F	/	4F	O	6F	o
10	DLE	Data Link Escape	^P	30	0	50	P	70	p
11	DC1	XON (Device control 1)	^Q	31	1	51	Q	71	q
12	DC2	Device control 2	^R	32	2	52	R	72	r
13	DC3	XOFF (Device control 3)	^S	33	3	53	S	73	s
14	DC4	Device control 4	^T	34	4	54	T	74	t
15	NAK	Negative acknowledgement	^U	35	5	55	U	75	u
16	SYN	Synchronous idle	^V	36	6	56	V	76	v
17	ETB	End of Transmission Block	^W	37	7	57	W	77	w
18	CAN	Cancel	^X	38	8	58	X	78	x
19	EM	End of Medium	^Y	39	9	59	Y	79	y
1A	SUB	Substitute	^Z	3A	:	5A	Z	7A	z
1B	ESC	Escape	^[	3B	;	5B	[	7B	{
1C	CuR	Cursor Right	^\	3C	<	5C	\	7C	|
1D	CuL	Cursor Left	^]	3D	=	5D	]	7D	}
1E	CuU	Cursor Up	^^	3E	>	5E	^	7E	~
1F	CuD	Cursor Down	^_	3F	?	5F	_	7F	del

Tableau 14 : Code ASCII standard (7 bits)
Les codes 20_h à $7E_h$ correspondent à des caractères imprimables.

- Les 32 premiers codes (00 à $1F_h$), et le dernier ($7F_h$) sont des codes de commandes.
  Certains sont des commandes d'imprimante (ou du curseur sur un écran utilisé en mode caractères).
  Par exemple :
    - BELL (= 07_h) : Beep = sonnette
    - CR (= $0D_h$) : Carriage Return[1] = retour en début de ligne
    - LF (= $0A_h$) : Line Feed = passage à la ligne = saut à la ligne suivante
    - FF (= $0C_h$) : Form Feed = passage à la page suivante
    - Home : home = envoyer le curseur au coin supérieur gauche
    - CuR : Cursor Right = déplacer le curseur d'un caractère vers la droite
    - CuL : Cursor Left = déplacer le curseur (d'un caractère) vers gauche
    - CuU : Cursor Up = déplacer le curseur (d'une ligne) vers le haut
    - CuD : Cursor Down = déplacer le curseur (d'une ligne) vers le bas

- Vous devez retenir les codes suivants par cœur :

Hex	Char	Comment
00	NULL	Null
0A	LF	Line Feed (passage à la ligne)
0C	FF	Form Feed (passage à la page)
0D	CR	Carriage Return (retour en début de ligne)
1B	ESC	Escape
11	Xon = Ctrl Q	Autorisation de continuer la transmission
13	Xoff = Ctrl S	Demande d'arrêter la transmission
41	A	Lettre A majuscule. Les autres majuscules suivent dans l'ordre.
55	U	Lettre U majuscule. Envoyer sur une ligne série une suite de caractères U majuscules (sans bit de parité) donne une onde carrée aisément reconnaissable à l'oscilloscope
61	a	Lettre a minuscule. Les autres minuscules suivent dans l'ordre.

Tableau 15 : Codes ASCII à connaître par coeur

[1] Historiquement : retour du chariot (d'une machine à écrire) en début de ligne

4.5 EXERCICES

20. Remplissez le tableau des codes Gray pour les nombres de 0 à 31
    Réponse :

21. Que faut-il faire pour continuer ce tableau, par exemple jusque N = 127 ?

22. Donnez les codes ASCII standard en 7 bits, de la chaîne de caractères "Hello 3"
    (attention, après le caractère "o", il y a un caractère blanc)

5 Systèmes combinatoires et fonctions combinatoires de base

5.1 Systèmes combinatoires. État d'entrée. État de sortie

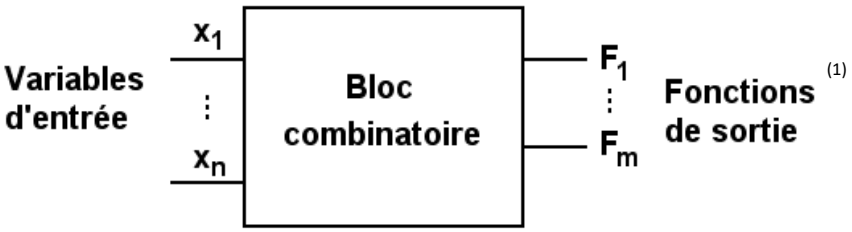


Figure 9 : Un système combinatoire avec n entrées et m sorties

- On appelle "état d'entrée" toute combinaison possible des valeurs (binaires) des variables d'entrée :
  - Un système à 1 entrée a 2 états d'entrées différents possibles (0 et 1)
  - Un système à n entrées a 2^n états d'entrées différents possibles (car chacune des n variables d'entrée possède 2 états).
- On appelle "état de sortie" toute combinaison possible des valeurs (binaires) des variables de sortie :
  - Un système à 1 sortie a 2 états de sortie différents possibles (0 et 1)
  - Un système à m sorties a 2^m états de sortie différents possibles

5.2 Définition : système "combinatoire". Description d'un système combinatoire

> Un système est combinatoire si,
> à tout instant, l'état de sortie du système
> dépend seulement de l'état d'entrée du système au même instant.

- On peut décrire tout système combinatoire par une table de définition ou **table de vérité** : cette table spécifie la fonction (l'état de sortie), pour chaque état d'entrée possible (exemples : voir pages suivantes)
- Par conséquent, un système combinatoire *n'a pas de mémoire*
- **Un système combinatoire** est un système dans lequel *la notion de temps n'existe pas !* On considère que modifier une entrée a un effet *instantané* sur les sorties : on néglige les délais de propagation des circuits, on les considère comme nuls.

  C'est évidemment une approximation. En pratique, lorsqu'on modifie une entrée, il suffit d'attendre quelques dizaines de ns (1 ns = 1 nanoseconde = 10^{-9} s) avant de mesurer les sorties.

(1) Même si les « m » sorties sont des « variables » booléennes, le fait d'utiliser « fonctions » de sortie met en évidence la notion de fonction combinatoire sur base des « n » entrées x_i.

5.3 Fonctions combinatoires (opérateurs ou portes logiques) de base

- On appelle "Porte « x »", "Opérateur « x »", "«x» Gate",
  le circuit combinatoire qui réalise la fonction « x ».
  Une "Gate" peut avoir une ou plusieurs entrées, mais elle a une et une seule sortie.

- On peut écrire toute fonction binaire, quel que soit le nombre de variables, en combinant quelques fonctions de base. Les 3 fonctions de bases usuelles sont : NOT, AND, OR

- Dans un schéma, on représente les portes les plus utilisées par un symbole normalisé, un "logigramme". Il existe plusieurs normes pour les logigrammes, dont :
  - Les normes américaines MIL. Ces normes sont utilisées par les fabricants américains, japonais, coréens … et dans ce syllabus.
  - Les normes de la Commission Électrotechnique Internationale (IEC). En Belgique, ces normes doivent être utilisées pour les documents légaux.
  - Les normes allemandes DIN40700.

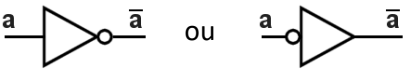
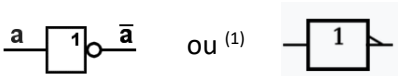
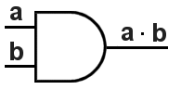
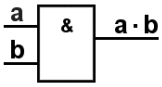
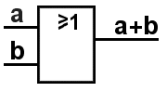
	Norme MIL	Norme IEC
NOT (inverseur)		
AND		
OR		

Tableau 17 : Quelques symboles suivant les normes MIL et IEC

5.4 Fonction NOT (l'inverseur)

- La fonction "NOT" (2) est une fonction d'une variable :
  la sortie est l'inverse, l'opposé, le complément de l'entrée .

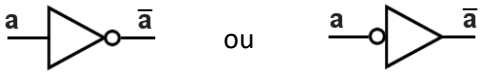


Figure 10 : Logigrammes MIL d'un inverseur (Fonction NOT)

Entrée a	Sortie $s = \bar{a}$
0	1
1	0

Tableau 18 : Fonction NOT : Table de vérité

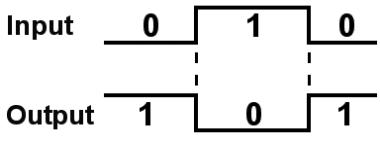


Figure 11 : Fonction NOT : chronogramme

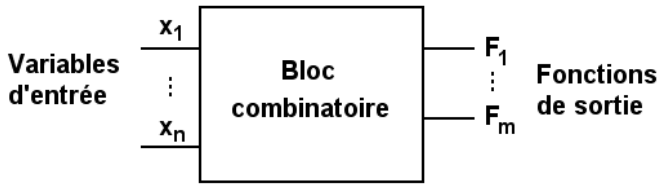
(1) IEC étant une association de standadisation fournissant des normes payantes, il est parfois difficile de trouver le symbole exact. On trouve les inversions représentées par une boule ou par un petit triangle. Si on se réfère à *https://en.wikibooks.org/wiki/Practical_Electronics/Logic_symbols* (consulté 26/07/23), l'inversion par un boule correspond à la norme British BS et l'inversion par un triangle correspond à l'inversion par la norme IEC.

(2) Certains livres français l'appellent fonction "NON"

5.5 Table de vérité

- Nous avons déjà rencontré 2 exemples de Table de vérité : la table de vérité de la fonction NOT (**Tableau 18**), et la table de vérité de la fonction AND (**Tableau 5** et page suivante), et nous avons dit qu'une "Table de Vérité" d'une fonction booléenne spécifie la valeur de la fonction (= sortie), pour chacune des combinaisons possibles des variables (= des entrées).

- Plus généralement, pour un système logique combinatoire à n entrées et m sorties :



Ref. Figure 9 : Un système combinatoire avec n entrées et m sorties

  - Les lignes de la Table de Vérité donnent la liste des combinaisons possibles des variables d'entrée (= variables indépendantes), sous forme d'*une ligne par état des entrées*.
    Avec n entrées, il y a 2^n états d'entrées différents $\Rightarrow$ la table a 2^n lignes
  - Chaque colonne de la table spécifie les valeurs d'une des sorties, pour chacun des états d'entrées (c.-à-d. pour chaque ligne)

Entrées					Sorties			
x_n	x_{n-1}		x_2	x_1	F_1	F_2		F_m
0	0	…	0	0	…	…	…	…
0	0	…	0	1	…	…	…	…
0	0	…	1	1	…	…	…	…
		…			…	…	…	…
1	1	…	1	1	…	…	…	…

Tableau 19 : Table de vérité de m fonctions de n variables

- Vocabulaire :
  - Les variables d'entrée sont aussi appelées variables « indépendantes »
  - Les sorties sont appelées « fonctions » ou « variables dépendantes »

- Remarques :
  - *Dans ce syllabus*, on range d'habitude les variables d'entrées avec *l'indice le plus faible dans la colonne de droite des entrées.*
    On sépare aussi la partie gauche (entrées) de la partie droite (sorties) par une ligne plus épaisse
  - En général, tous les auteurs rangent les lignes de la table par ordre croissant (en commençant par tous des "0" dans la première ligne, et en terminant par tous des "1" dans la dernière ligne).

5.6 Fonction BUFFER

- Le "Buffer" est une fonction d'une variable très simple : la sortie est égale à l'entrée

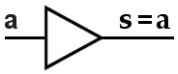


Figure 12 : Logigrammes MIL de la Fonction Buffer

Entrée a	Sortie s = a
0	0
1	1

Tableau 20 : Buffer : Table de vérité

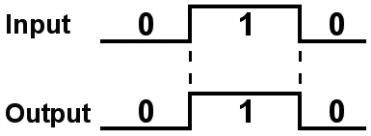
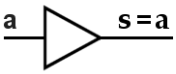


Figure 13 : Fonction NOT : chronogramme

- Le "Buffer" n'est pas utile au point de vue logique booléenne;
  il est utile dans de nombreux cas au point de vue électrique, comme amplificateur,
  par ex. à l'entrée ou à la sortie de cartes électroniques.

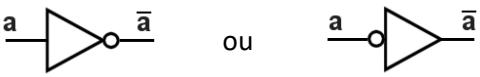
5.7 Dessin : Boule (bille, cercle) d'inversion

- Comparez les dessins



Ref. Figure 12 : Logigrammes MIL de la Fonction Buffer

et



Ref. Figure 10 : Logigrammes MIL d'un inverseur (Fonction NOT)

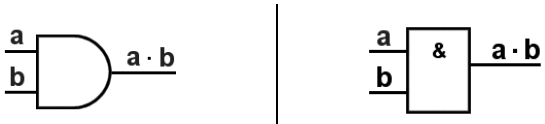
En fait, dans cette **Figure 10,** la bille représente l'inversion.

Les deux dessins de l'inverseur sont équivalents, on les utilise tous les deux
(celui de gauche sans doute un peu plus souvent).

En anglais, la bille d'inversion se dit : " bubble " (ou parfois : "bobble")

5.8 Fonction AND

- La fonction "AND" est une fonction de 2 variables (ou plus),
  qui correspond à l'utilisation du mot "ET" en français courant :
  - Si la variable a ET la variable b valent 1 , la fonction AND vaut 1
  - Si au moins une variable d'entrée est à 0 , la sortie vaut 0



Norme MIL | **Norme IEC**

Figure 14 : Logigrammes de la fonction AND

Entrées		Sortie
b	a	s = a . b
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 21 : Table de vérité de la fonction AND

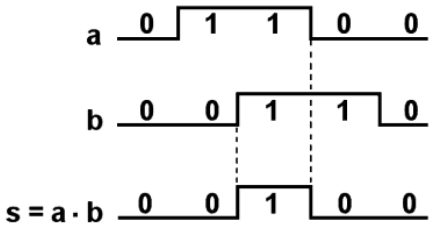
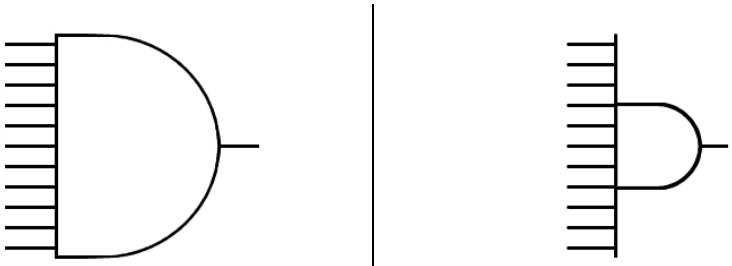


Figure 15 : Exemple de chronogramme pour une porte AND [(1)]

- Le AND est souvent appelé "**PRODUIT logique**", car nous verrons que de nombreuses propriétés du AND ressemblent à celles du produit des nombres réels, en algèbre.

- On peut généraliser la fonction AND à un nombre quelconque de variables ;
  - Si toutes les variables d'entrée valent 1, la fonction AND vaut 1
  - dès qu'une variable d'entrée est à 0 , la fonction AND vaut 0



Déconseillé (lourd) | Conseillé

Figure 16 : Logigramme d'un AND à plus de 2 entrées
(Dessin MIL ; même principe pour le dessin IEC)

[(1)] Le Tableau 21 est une table de vérité : l'ensemble des états possibles des entrées, et la valeur correspondante de la sortie. Les lignes y sont cotées dans un ordre usuel, et il n'y a pas de notion de temps dans une table de vérité.
La Figure 15 montre les mêmes informations qu'au Tableau 21 , mais la Figure 15 est un chronogramme.
L'axe horizontal de la Figure 15 est le temps ; il se fait que dans cet exemple de chronogramme,
a succession des états d'entrées de la porte AND n'est pas dans l'ordre des lignes du Tableau 21.

5.9 Fonction OR (logical OR)

- La fonction OR (Logical OR, Inclusive OR), en français : "OU", "OU logique", "OU inclusif", "somme logique", correspond à l'utilisation du mot "OU" en langage courant :
  - Si au moins une des entrées vaut 1, la sortie vaut 1
  - Si les 2 entrées valent 0 , la sortie vaut 0

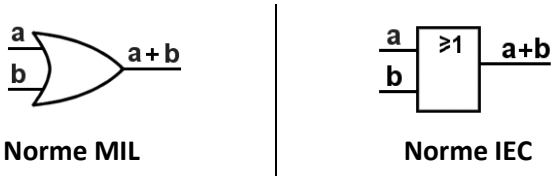


Figure 17 : Fonction OR

Entrées		Sortie
b	a	s = a + b
0	0	0
0	1	1
1	0	1
1	1	1

Tableau 22 : Table de vérité de la fonction OR

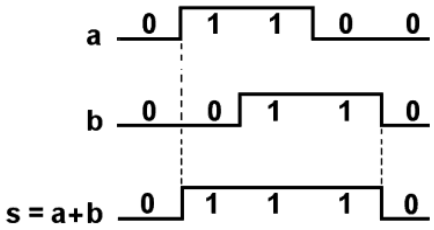


Figure 18 : Exemple de chronogramme pour une porte OR [(1)]

- Le OR est souvent appelé "**SOMME logique**" car certaines propriétés du OR ont une ressemblance <u>partielle</u> avec celles de la somme (l'addition) en algèbre des réels.

- On peut généraliser la fonction OR à un nombre quelconque de variables :
  - si toutes les variables d'entrée valent 0, la fonction OR vaut 0
  - si au moins une des entrées vaut 1, la fonction OR vaut 1

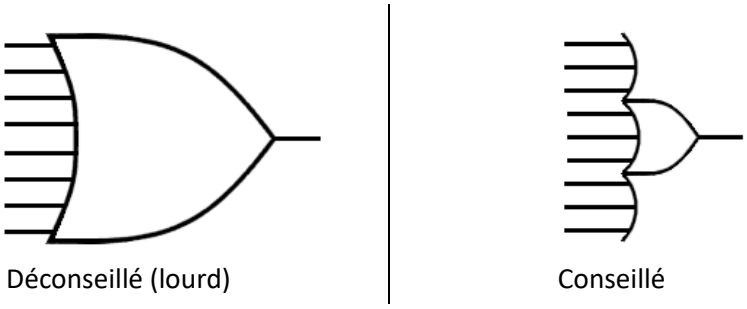
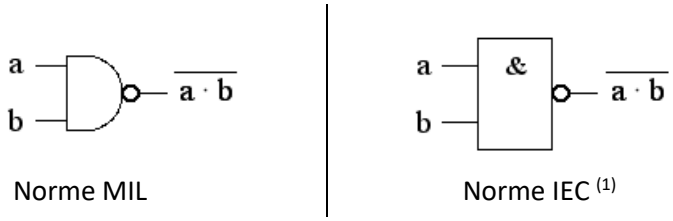


Figure 19 : Logigramme d'un OR à plus de 2 entrées
(Dessin MIL; même principe pour le dessin IEC)

[(1)] Même remarque que page précédente

5.10 Fonction NAND

- Le NAND est un AND suivi d'un NOT
- Propriétés du NAND :
  - Si toutes les entrées sont à 1 , la fonction NAND vaut 0
  - Si au moins une entrée est à 0 , la fonction NAND vaut 1



Norme MIL | Norme IEC [1]

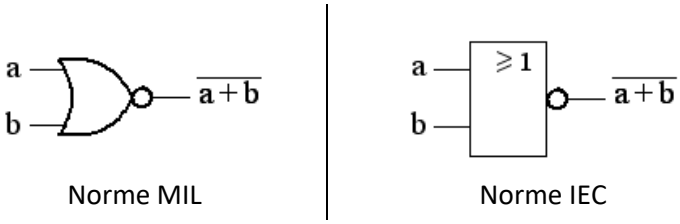
Figure 20 : Logigrammes de la fonction NAND (à 2 variables)

a	b	$s = \overline{a \cdot b}$
0	0	1
0	1	1
1	0	1
1	1	0

Tableau 23 : Table de vérité de la fonction NAND (à 2 variables)

5.11 Fonction NOR

- Le NOR est un OR suivi d'une inversion
- Propriétés du NOR :
  - Si toutes les entrées sont à 0 , la sortie vaut 1
  - Si au moins une des variables est à 1 , la sortie vaut 0



Norme MIL | Norme IEC

Figure 21 : Logigrammes de la fonction NOR (à 2 variables)

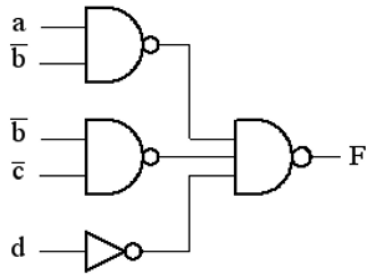
a	b	$s = \overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

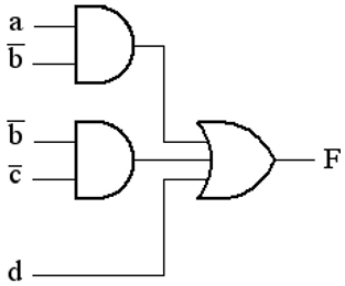
Tableau 24 : Table de vérité de la fonction NOR (à 2 variables)

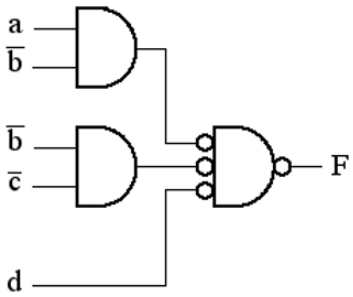
[1] Comme déjà mentionné au tableau 17 page 37, la fonction d'inversion pour les symboles logiques européens est représentée parfois par une boule, parfois par un triangle ; ces deux façons de faire sont acceptées.

5.12 EXERCICES

23. Donnez les équations correspondant aux schémas ci-dessous







24. Dessinez les schémas correspondant aux équations ci-dessous

$F = a.\overline{b} + \overline{b}.\overline{c} + d$

$F = \overline{\overline{a \cdot \overline{b}} \; . \; \overline{\overline{b}.\overline{c}} \; . \; \overline{d}}$

6 Algèbre de Boole : Propriétés des opérateurs booléens NOT, AND, OR

- L'algèbre logique binaire (l'algèbre de Boole) permet de décrire les systèmes à variables binaires. Ces systèmes utilisent les opérateurs NOT, AND, et OR.
- Notations : Au lieu de "AND" on écrit souvent "×", ou "." ou "·"
  Au lieu de "OR" on écrit souvent "+"
- **Convention** : Précédence (priorité) des opérateurs :
  - " . " a une précédence plus élevée que " + "
    (l'opérateur AND a une priorité plus élevée que l'opérateur OR)
    … autrement dit : on effectue d'abord les AND, et ensuite les OR
  - Au besoin, on utilise des parenthèses pour préciser / forcer l'ordre des opérations :

  Exemples :
  - z = a + b . c signifie : z = a + (b . c) et est différent de (a + b) . c
  - s = [(a+b) . c + a . d] . e est équivalant à l'ensemble des équations :
    - u = a+b
    - v_1 = u . c
    - v_2 = a . d
    - x = v_1 + v_2
    - s = x . e

6.1 Propriétés inspirées des opérateurs algébriques + et × pour les réels

- Les propriétés des opérateurs AND et OR *ressemblent* à celles des opérateurs algébriques de multiplication et d'addition × et + pour les nombres réels,
  mais avec des adaptations dues au fait que les variables sont binaires :

Propriété	AND	OR
Commutativité [(1)]	a . b = b . a	a + b = b + a
Associativité [(2)]	(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)
Idempotence	a . a . … . a = a	a + a + … + a = a
Existence d'un Neutre	a . 1 = 1 . a = a	a + 0 = 0 + a = a
Existence d'un Absorbant	a . 0 = 0 . a = 0	a + 1 = 1 + a = 1
Complémentation	a . $\overline{a}$ = 0	a + $\overline{a}$ = 1

Tableau 25 : Propriétés de base des opérateurs booléens NOT AND OR

[(1)] Commutativité : signifie que l'ordre des opérandes a et b n'a pas d'importance

[(2)] Associativité : l'ordre de 2 opérations AND n'a pas d'importance : on peut associer les opérandes a, b, c dans l'ordre qu'on veut. Définition : a . b . c = (a .b) . c. Propriété d'associativité ⇒ a . b . c = a . (b . c)

6.2 Théorème de "De Morgan"

6.2.1 Théorème de De Morgan à 2 variables

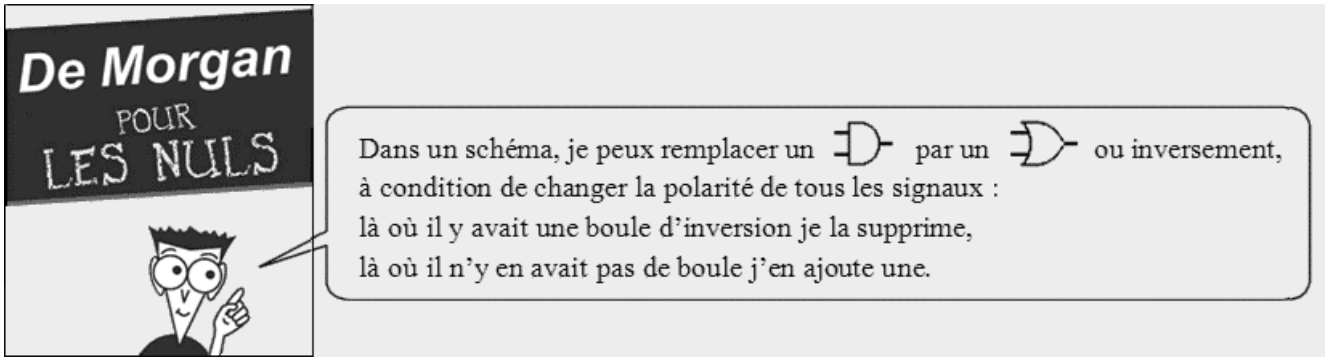
- Auguste De Morgan, 1806 – 1871, http://en.wikipedia.org/wiki/Augustus_de_Morgan

> On peut remplacer un OR par un AND
>
> ou inversement,
>
> à condition d'inverser les entrées, ainsi que la sortie.

Ce qui s'écrit :

$$\overline{a+b} = \overline{a} \cdot \overline{b} \quad (1) \qquad \text{et} \qquad \overline{a \cdot b} = \overline{a} + \overline{b} \quad (2)$$

- Ce théorème important est très souvent utilisé. Vous devez savoir l'utiliser sans hésiter.
  Mais plutôt que de retenir des formules d'aspect mathématique
  (difficiles à lire dès que plusieurs barres d'inversion sont superposées),
  on peut utiliser un procédé graphique, très simple :



Exemples :

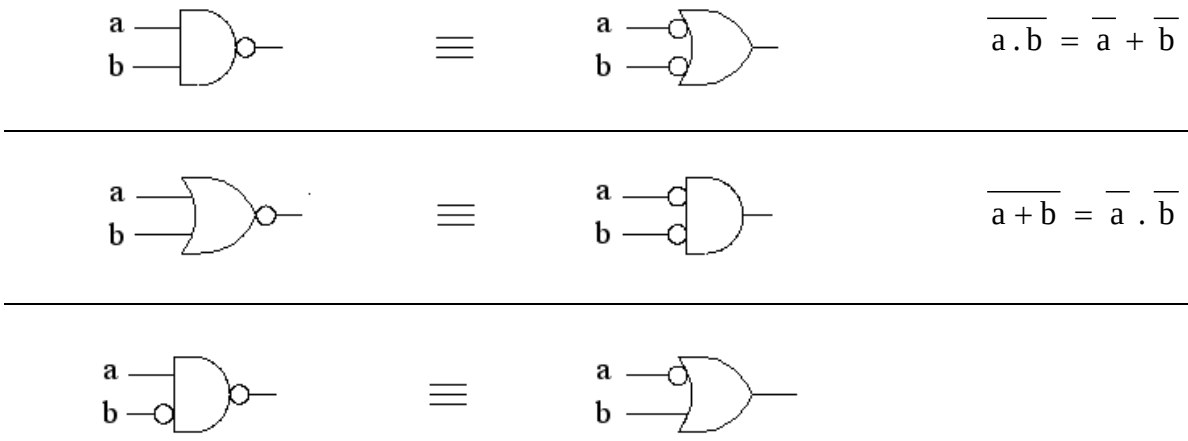


Figure 22 : Appliquer De Morgan *dans un schéma* est très facile

(1) Souvent appelé aussi " 1er théroème de De Mogan " , ou " 1ère loi de De Morgan "

(2) Souvent appelé aussi " 2ème théorème de De Mogan" , ou " 2ème loi de De Morgan "

6.2.2 Théorème de De Morgan à n variables

- Le théorème de De Morgan est généralisable à n variables :

$$\overline{a \cdot b \cdot c \cdot \ldots \cdot n} = \overline{a} + \overline{b} + \overline{c} + \ldots + \overline{n}$$

$$\overline{a + b + c + \ldots + n} = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \ldots \cdot \overline{n}$$

6.3 EXERCICES

25. Écrivez une expression booléenne qui vaut 1
quand au moins une des quatre variables A, B, C, D vaut 1.

26. Écrivez une expression booléenne qui vaut 1
quand toutes les quatre variables A, B, C, D valent 1.

27. Considérez chaque propriété citée dans le **Tableau 25**
cette propriété est-elle vraie en algèbre classique avec des variables nombres réels ?

Propriété	**Opérateur algébrique**			
	Multiplication	vrai/ faux	**Addition**	vrai/ faux
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)	
Idempotence	a . a . … . a = a		a + a + … + a = a	
Existence d'un Neutre	a . 1 = 1 . a = a		a + 0 = 0 + a = a	
Existence d'un Absorbant	a . 0 = 0 . a = 0		a + k = k + a = k	
Complémentation	a . $\overline{a}$ = 0		a + $\overline{a}$ = 1	

Réf. Tableau 25 : Propriétés de base des opérateurs booléens NOT AND OR
Validité ou non-validité de ces propriétés lorsque les variables a, b, c , k sont des réels

28. Simplifiez les équations suivantes :

a + a = …

a . a = …

a + $\overline{a}$ = …

a . $\overline{a}$ = …

a + a . b = …

a + $\overline{a}$. b = …

a . b + a . $\overline{b}$ = …

a . ($\overline{b+c}$) = …

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

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

29. Dressez la table de vérité de la fonction s = a . $\overline{(b+c)}$. Simplifiez s

30. Dressez la table de vérité de la fonction s = (a+b).c + b . Simplifiez s

31. Dressez la table de vérité de la fonction $s = (a+b.c)\,(\overline{b}+\overline{c})$. Simplifiez s

32. Démontrez le théorème de De Morgan $\overline{a \,.\, b} = \overline{a} + \overline{b}$ en remplissant une table de vérité

33. Appliquez le théorème de De Morgan à l'expression : s = $\overline{a} + b$

34. Appliquez le théorème de De Morgan à l'expression : s = a . ($\overline{b + c}$)

35. Appliquez le théorème de De Morgan à l'expression : s = E . ($\overline{\beta + \eta}$) + β

36. Trouvez la forme minimale de l'équation suivante et réalisez le schéma de câblage en utilisant seulement des portes OR : $S = (A + \overline{B + C}).(A + \overline{\bar{B}.\bar{C}}) + \overline{A.B}.(\bar{C}.D + C.D)$

7 Algèbre de Boole : Synthèse d'une fonction

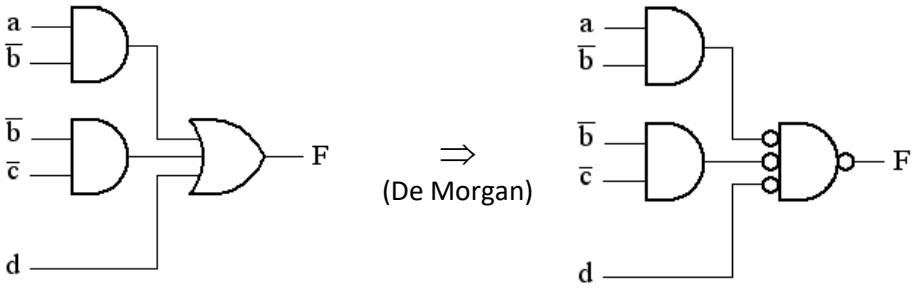
7.1 Expressions et systèmes équivalents

- On a vu que tout système combinatoire peut être décrit par une table de vérité.
  Cette table définit la fonction du système :
  elle donne la valeur de la sortie pour toutes les combinaisons des valeurs des entrées.
  On dit que le système combinatoire "réalise" cette fonction.

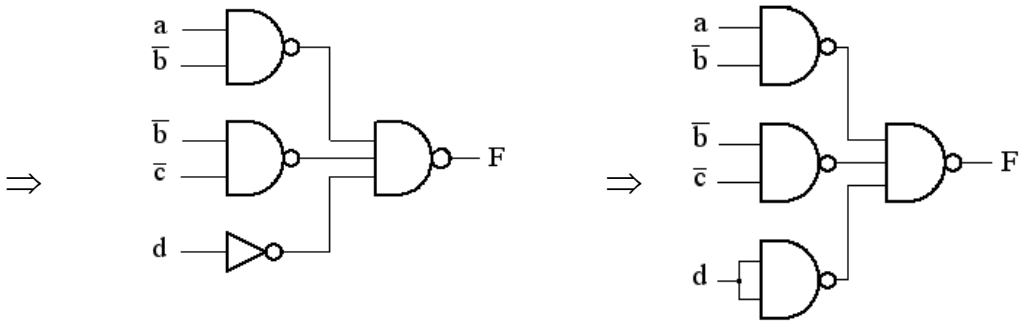
- Si 2 systèmes combinatoires différents réalisent la même fonction combinatoire,
  on dit que ces deux systèmes sont (fonctionnellement) *équivalents* .

- Deux expressions algébriques qui sont égales réalisent aussi la même fonction,
  et les logigrammes correspondants, même s'ils sont différents, sont équivalents.

- Exemple : $F = a.\bar{b} + \bar{b}.\bar{c} + d = \overline{\overline{a \cdot \bar{b}} \; . \; \overline{\bar{b}.\bar{c}} \; . \; \bar{d}}$



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



$F = \overline{\overline{a \cdot \bar{b}} \; . \; \overline{\bar{b}.\bar{c}} \; . \; \bar{d}}$ $F = \overline{\overline{a \cdot \bar{b}} \; . \; \overline{\bar{b}.\bar{c}} \; . \; \bar{d}}$

Figure 23 : Deux circuits équivalents, l'un en AND–OR, l'autre avec des NAND

- Remarquez qu'on a remplacé
  - un circuit AND-OR (schéma en haut à gauche)
  - par un circuit uniquement réalisé avec des NAND, qui a exactement les mêmes variables d'entrée, la même variable de sortie, et la même fonction.

  Ce remplacement est toujours possible, car le NAND est un opérateur complet
  (voir page suivante).

7.2 Opérateurs complets

7.2.1 Ensemble d'opérateurs complet. Opérateur complet

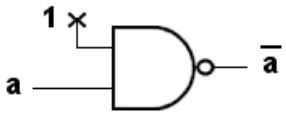
- On appelle "ensemble d'opérateurs complet", un ensemble d'opérateurs permettant de réaliser n'importe quelle fonction combinatoire.
- On appelle "opérateur complet", un opérateur permettant (à lui seul) de réaliser (synthétiser) n'importe quelle fonction combinatoire.

7.2.2 Les trois opérateurs NOT, AND et OR forment un ensemble d'opérateurs complet

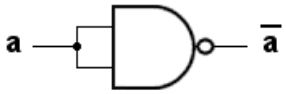
- En effet, il existe une méthode de synthèse de n'importe quelle fonction au moyen de ces 3 opérateurs (voir § 7.6 SDP standard d'une fonction combinatoire)

7.2.3 Le NAND est un opérateur complet

- En effet : le NAND permet de réaliser la fonction NOT :



En équations : $1 . a = a$ (neutre) $\Rightarrow \overline{1.a} = \bar{a}$



En équations : $a . a = a$ (idempotence) $\Rightarrow \overline{a.a} = \bar{a}$

Figure 24 : Remplacement d'un NOT par un NAND

Note: ces 2 schémas sont équivalents logiquement, mais électriquement il y a une nuance :

- dans le 1er schéma, on ne charge qu'une fois le circuit fournissant la variable a .
- dans le 2nd schéma, on le charge 2 fois.

- D'autre part, le NAND permet de réaliser la fonction AND

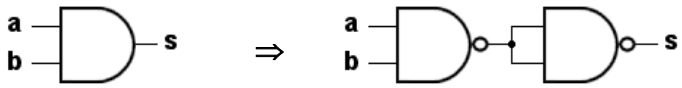


Figure 25 : Remplacement d'un AND par des NAND

- Enfin, le NAND permet de réaliser la fonction OR

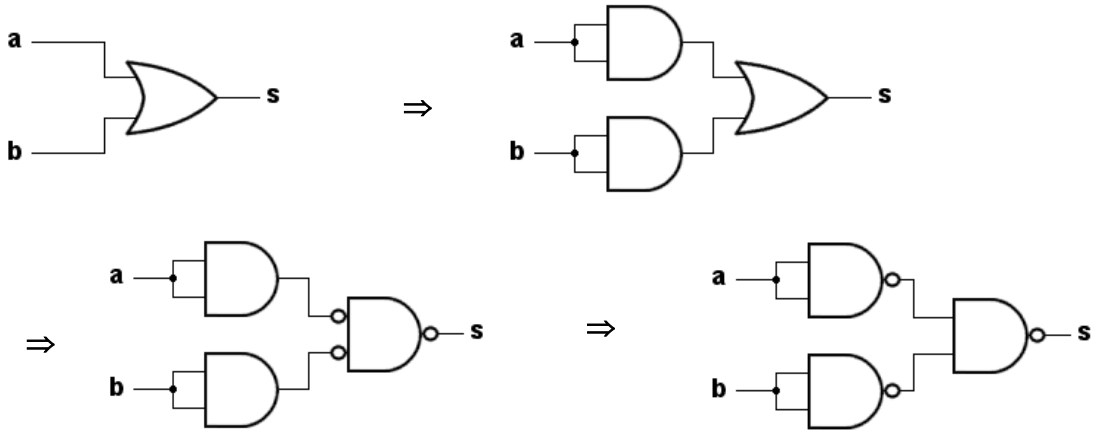


Figure 26 : Remplacement d'un OR par des NAND

⇒ Conclusion : le NAND est un opérateur complet.

7.3 Numérotation usuelle des lignes d'une table de vérité

- Dans une table de vérité, on écrit dans les colonnes de gauche
  l'ensemble de toutes les valeurs possibles des entrées
  (une ligne correspond à une valeur particulière de l'ensemble des entrées).

- En général, on ordonne les états d'entrée (les lignes de la table)
  par ordre croissant du nombre binaire formé par les bits d'entrées

- A gauche de la table, on peut ajouter une colonne contenant les "numéros d'état d'entrée"
  (numéros qu'on écrit d'habitude en décimal).
  Exemple: pour un système combinatoire à 3 entrées,
  l'état d'entrée $cba = 110$ sera numéroté "$6_{déc}$"

n° état $n = ba_{déc}$	Entrées b	Entrées a	Sortie S
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1

Tableau 27 : Fonction logique S à 2 entrées (poids fort: b, poids faible: a)

$n = cba_{déc}$	Entrées c	Entrées b	Entrées a	Sortie F
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Tableau 28 : Fonction logique F à 3 entrées (poids le plus fort: c, poids le plus faible: a)

- Remarque:
  Numéroter les états d'entrée suppose qu'on a choisi une *pondération pour les variables*
  c.-à-d. qu'on a spécifié *l'ordre dans lequel on range* les variables d'entrée
  pour constituer le numéro d'état.

  Exemple : pour la fonction F du **Tableau 29** : cette fonction F a 3 entrées binaires

  - si on spécifie [c = poids fort, b = poids intermédiaire, a = poids faible] ,
    alors l'état d'entrée $c = 0,\ b = 1,\ a = 1$
    est numéroté : $n = cba_{bin} = 011_{bin} = 3_{déc}$

  - mais si on spécifiait [a = poids fort, b = poids intermédiaire, c = poids faible] ,
    alors ce même état d'entrée $c = 0,\ b = 1,\ a = 1$
    serait numéroté : $n = abc_{bin} = 110_{bin} = 6_{déc}$

7.4 Monôme, polynôme. Minterm

- **Définition** (inspirée de l'algèbre pour les nombres réels) :

> Un **monôme** est un produit logique (AND) de plusieurs variables (chacune des variables peut être complémentée, ou non) .
>
> Une variable seule, complémentée ou non, est aussi considérée comme un monôme.

Exemples :

- $a \cdot c$ est un monôme
- $\overline{a} \cdot b \cdot \overline{d}$ est un monôme
- d est un monôme
- $\overline{b}$ est un monôme

- **Définition** (inspirée de l'algèbre pour les nombres réels) :

> Un **polynôme** est une somme logique (OR) de monômes.

Exemples :

- $d + \overline{a} \cdot b \cdot \overline{d}$ est un polynôme
- $c + a \cdot \overline{b}$ est un polynôme

- **Définition**

> Soit une fonction logique booléenne qui a n variables d'entrée :
> Un **minterm** est un monôme (un AND) qui **inclut toutes les variables d'entrée** (chaque variable est complémentée ou non).

Exemples :

- pour un système combinatoire à 2 entrées a, b : $\overline{b} \cdot a$ est un minterm
- pour un système combinatoire à 3 entrées a, b, c : $\overline{b} \cdot a$ *n'est pas* un minterm
  $c \cdot \overline{b} \cdot a$ est un minterm
  $c \cdot b \cdot a$ est un autre minterm

- **Nombre de minterms d'un système à n entrées** :
  - Un système avec n entrées a **2^n minterms différents**
    (car chaque variable peut y apparaître sous 2 formes : vraie ou complémentée).

- **Numérotation des minterms** : on note les minterms : K_i $i = 0, 1, 2, \ldots, 2^n - 1$

- **Pour construire K_i** (voyez les exemples page suivante) :
  On considère le nombre i formé par les variables binaires d'entrées :
  - si une variable d'entrée vaut 1 dans i , on écrit cette variable sous forme vraie
  - si une variable d'entrée vaut 0 dans i , on l'écrit sous forme complémentée

- **Propriété : l'indice i est le n° de l'état (du seul état) d'entrée pour lequel K_i vaut 1**
  En effet, un minterm étant réalisé par un AND, ne vaut 1 que
  - si toutes les variables écrites sous forme vraie valent 1, et
  - si toutes les variables complémentées valent 0 .

7.5 Exemples

- Exemple 1 : Système combinatoire à 2 entrées b, a (poids fort : b, poids faible : a)
  - 4 minterms : $K_0 = \overline{b} \cdot \overline{a}$, $K_1 = \overline{b} \cdot a$, $K_2 = b \cdot \overline{a}$, $K_3 = b \cdot a$
  - Voici les valeurs de ces 4 minterms, pour chacun des états d'entrée :

n° d'état	entrées		minterms			
$ba_{déc}$	b	a	$K_0 = \overline{b} \cdot \overline{a}$	$K_1 = \overline{b} \cdot a$	$K_2 = b \cdot \overline{a}$	$K_3 = b \cdot a$
0	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
2	1	0	0	0	**1**	0
3	1	1	0	0	0	**1**

Tableau 29 : Valeurs des 4 minterms d'un système combinatoire à 2 entrées

- Exemple 2 : Système combinatoire à 3 entrées c, b, a
  (poids le plus fort : c, poids le plus faible : a)
  - 8 minterms: $K_0 = \overline{c} \cdot \overline{b} \cdot \overline{a}$, $K_1 = \overline{c} \cdot \overline{b} \cdot a$, $K_2 = \overline{c} \cdot b \cdot \overline{a}$, $K_3 = \overline{c} \cdot b \cdot a$,

    $K_4 = c \cdot \overline{b} \cdot \overline{a}$, $K_5 = c \cdot \overline{b} \cdot a$, $K_6 = c \cdot b \cdot \overline{a}$, $K_7 = c \cdot b \cdot a$
  - Voici les valeurs de ces 8 minterms, pour chacun des 8 états d'entrée :

n° d'état	entrées			minterms							
$cba_{déc}$	c	b	a	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	0	1	0	**1**	0	0	0	0	0	0
2	0	1	0	0	0	**1**	0	0	0	0	0
3	0	1	1	0	0	0	**1**	0	0	0	0
4	1	0	0	0	0	0	0	**1**	0	0	0
5	1	0	1	0	0	0	0	0	**1**	0	0
6	1	1	0	0	0	0	0	0	0	**1**	0
7	1	1	1	0	0	0	0	0	0	0	**1**

Tableau 30 : Valeurs des 8 minterms d'un système combinatoire à 3 entrées

- Exemple 3 : Système combinatoire à 4 variables d'entrées d, c, b, a
  (poids le plus fort: d, poids le plus faible: a).
  - Le minterm associé à l'état d'entrée $dcba = 10_{déc} = 1010_{bin}$
    est $K_{10} = d \cdot \overline{c} \cdot b \cdot \overline{a}$
    Si d = 1, c = 0, b = 1, et a = 0 alors $K_{10} = 1$

7.6 SDP standard d'une fonction combinatoire

- La propriété des minterms de valoir 1 pour un et un seul état d'entrée
  fournit la fonction combinatoire correspondant à la table de vérité :

Toute fonction combinatoire est égale à la somme logique (OR)
des minterms associés aux états d'entrée pour lesquels la fonction vaut 1 .
Cette expression est appelée "SDP standard" de la fonction[1] .

Trouver la SDP standard d'une fonction est très simple :

- Exemple: on donne la fonction de 2 variables s (a, b). On demande sa SDP standard.

n°	entrées b	a	sortie f		minterms
0	0	0	1	←	$K_0 = \overline{b} \cdot \overline{a}$
1	0	1	1	←	$K_1 = \overline{b} \cdot a$
2	1	0	0		$K_2 = b \cdot \overline{a}$
3	1	1	0		$K_3 = b \cdot a$

Tableau 31 : Recherche de la SDP standard d'une fonction f
A droite de la table de vérité, on a ajouté les minterms

Solution :

- On cherche les 1 dans la colonne f (on les a marqués avec deux flèches)
- D'où : SDP standard : $f = K_0 + K_1 \Rightarrow f = \overline{b}.\overline{a} + \overline{b}.a$

- Note : obtenir la SDP standard à partir de la table de vérité est donc très simple, mais il existe quasi toujours un résultat (un polynôme) plus simple pour la fonction (ici c'est $s = \overline{b}$). Pour trouver ce résultat simple : voir chapitre : Méthode de Karnaugh.

- Note : pour cette même fonction, si on choisit un ordre différent pour les colonnes des variables d'entrée : on écrit la table de vérité dans un autre ordre, et les minterms changent de lignes

n°	entrées a	b	sortie f		minterms
0	0	0	1	←	$K_0 = \overline{a} \cdot \overline{b}$
1	0	1	0		$K_1 = \overline{a} \cdot b$
2	1	0	1	←	$K_2 = a \cdot \overline{b}$
3	1	1	0		$K_3 = a \cdot b$

mais on arrive bien au même résultat final : SDP standard : $f = K_0 + K_2 \Rightarrow f = \overline{a}.\overline{b} + a.\overline{b}$

- Pour info : notation compacte de la SDP standard :
on écrit seulement les numéros des minterms pour lesquels la fonction vaut 1 :
au lieu d'écrire $F = K_2 + K_5 + K_6$, on écrit $F = \sum 2, 5, 6$

[1] "SDP" = Somme de Produits ; "standard", parce que les produits contiennent toutes les variables.
On dit aussi : "forme canonique en Somme De Produits", ou "forme canonique disjonctive"

7.7 EXERCICES

37. Montrez que le NOR est un opérateur complet

38. Trouvez la SDP standard de la fonction OR (OU inclusif)

39. Trouvez la SDP standard de la fonction NOR (OU inclusif)

Solution :

40. Cherchez la SDP standard de F :

n°	c	b	a	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Tableau 34 : Table de vérité d'une fonction de 3 variables

41. On désire visualiser sur un display les 10 chiffres décimaux codés en BCD.

    L'afficheur est un afficheur à cathode commune, qui est connectée au GND ; a, b, …, ou g = 1 allumera le segment correspondant).

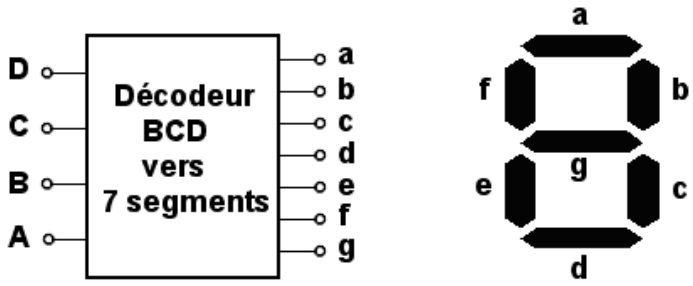


Figure 30 : Décodeur BCD vers 7 segments

Les différentes valeurs des entrées D C B A doivent provoquer l'allumage des segments dessinés dans le tableau ci-contre :

Pour l'exercice, on demande la SDP de la fonction a (la fonction qui allume le segment a).

Pour les 6 combinaisons de D C B A qui ne se présentent pas dans le code BCD, on imposera que la sortie a vaut 0

DCBA	segments allumés
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Figure 31 : Segments allumés

Note : le 74HC(T)4511 contient un quadruple latch (voir cours Electronique Numérique Partie 2, Systèmes Séquentiels), ainsi qu'un décodeur, et permet de piloter un afficheur 7 segments à cathode commune à partir d'un nombre de 4 bits.

8 Quelques fonctions combinatoires usuelles

8.1 Signal actif haut, signal actif bas. Choix du nom d'un signal

- On définit que tout signal binaire a 2 valeurs possibles : "actif", "inactif"
- Pour chaque signal indépendamment, on spécifie *la polarité du signal* :
  c.-à-d. le *niveau* logique et électrique qui correspond à l'état *actif* :
  - signal "actif haut" : l'état logique actif est 1 , et la tension est V_H
  - signal "actif bas" : l'état logique actif est 0 , et la tension est V_L

    Dans ce cas, au-dessus du nom du signal, on écrit une barre d'inversion
    et dans un schéma logique, on dessine un rond d'inversion
- Le choix des noms des signaux est vital pour faciliter la compréhension d'un schéma :
  Pour tout signal important, choisissez *un nom qui indique clairement le rôle de ce signal*
- Exemple : soit un signal qui spécifie qu'on détecte une erreur, ou que tout va bien.
  - Supposons que le signal au niveau *haut* signifie l'état d'erreur
    $\Rightarrow$ on pourrait par exemple appeler le signal comme ceci :
    - ERROR
    - $\overline{\text{OK}}$
    - $\overline{\text{OK}}$ /ERROR
    - ERROR/ $\overline{\text{OK}}$

    Ces 4 noms sont raisonnables, et sont équivalents: ils expriment la même chose.
    Le 4ème nom, ERROR/ $\overline{\text{OK}}$ est sans doute le plus facile à comprendre
  - Supposons au contraire que le niveau *bas* du signal signifie l'état d'erreur
    $\Rightarrow$ on pourrait par exemple appeler le signal comme ceci :
    - $\overline{\text{ERROR}}$
    - OK
    - OK/ $\overline{\text{ERROR}}$
    - $\overline{\text{ERROR}}$ /OK

    Ces 4 noms sont raisonnables, et sont équivalents: ils expriment la même chose.
    Le 3ème nom, OK/ $\overline{\text{ERROR}}$ est sans doute le plus explicite et facile à comprendre.
- La notion de signal "actif au niveau haut" ou "actif au niveau bas"
  permet de simplifier, d'unifier la description de certaines fonctions logiques

  Exemples : les circuits intégrés décodeurs 74xx139 et 74xx138 (voir page suivante).

8.2 Le décodeur $n \rightarrow 2^n$

8.2.1 Définition et fonction

- Entrées / Sorties
  - 2^n sorties, numérotées 0, 1, … , $2^n - 1$ (note : on commence au numéro 0)
  - n entrées : les "bits d'adresse"
    notés a, b, c, d, … (poids faible : a), ou notés $a_0, a_1, a_2, \ldots$ (poids faible : a_0)

    Sur ces entrées on applique un nombre binaire qui spécifie l'adresse d'une des sorties (= le n° de la sortie qu'on veut activer).

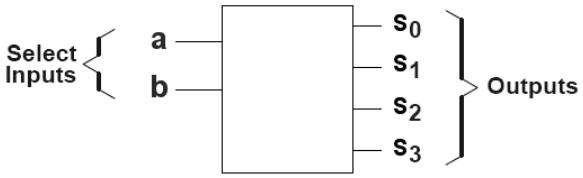


Figure 32 : Décodeur $2 \rightarrow 4$
n = 2 , 4 sorties numérotées 0 à 3

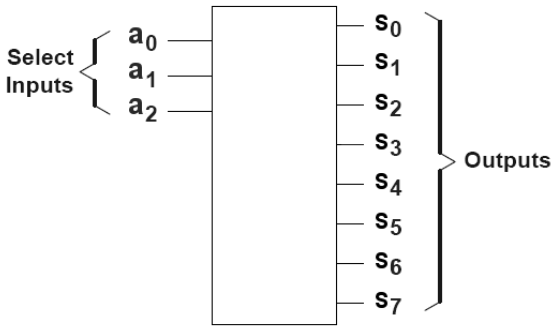


Figure 33 : Décodeur $3 \rightarrow 8$
n = 3 , 8 sorties numérotées 0 à 7

- Fonction du décodeur
  - le décodeur active une seule sortie: celle dont l'adresse est appliquée aux entrées
  - le décodeur désactive les autres sorties
- Équations des sorties : elles sont très simples : voir : **24 Annexe** (1)
- Exemple : décodeur $3 \rightarrow 8$ avec sorties actives hautes :
  - L'adresse est le nombre cba_{bin} (3 bits car $2^3 = 8$). Les 8 sorties sont $S_0, S_1, \ldots, S_7$

    Si par exemple on applique aux entrées $cba_{bin} = 110_{bin} = 6_{déc}$

    $\Rightarrow S_0 = S_1 = S_2 = S_3 = S_4 = S_5 = 0, \quad S_6 = 1, \quad S_7 = 0$
- Généralisation à un nombre quelconque de bits d'adresse :
  - Pour 2 sorties, il faut 1 bit d'adresse
  - Pour 4 sorties, il faut 2 bits d'adresses
  - Pour 8 sorties, il faut 3 bits d'adresses
  - . . .

8.2.2 74xx139 (dual $2 \rightarrow 4$ decoder), 74xx138 ($3 \rightarrow 8$ decoder), 74xx238 ($3 \rightarrow 8$ decoder)

- Téléchargez, imprimez et lisez et comprenez en détail (2) les datasheets de ces trois C.I. :
  - Le '139 contient deux décodeur $2 \rightarrow 4$, les '138 et '238 un décodeur $3 \rightarrow 8$
  - *Notez la polarité (active basse ou active haute) des sorties de ces différents chips !*

---

(1) C'est volontairement que nous ne les donnons pas ici. Nous voulons *utiliser* le circuit, donc comprendre ce qu'il fait, sa fonction, vue de l'extérieur : bref comprendre sa datasheet. L'utilisateur ne peut *pas* se baser sur une connaissance (supposée) de l'intérieur.

(2) Faites-le vraiment ! Apprendre seulement de la théorie ne sert à rien si on ne sait pas utiliser les chips !

8.3 Le démultiplexeur $1 \rightarrow 2^n$

8.3.1 Définition et fonction

- Entrées / Sorties
  - 1 entrée notée e,
  - n entrées "bits d'adresse" (notés a, b, c, d, ... Exemple ci-dessous : n = 2).
    Sur ces entrées on applique un nombre binaire qui spécifie l'adresse d'une des 2^n sorties (= le n° d'une des sorties).
  - 2^n sorties, numérotées 0, 1, ..., $2^n - 1$ (on commence à numéroter à 0)

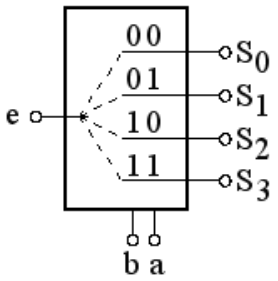


Figure 34 : Démultiplexeur $1 \rightarrow 4$. Schéma fonctionnel
n = 2. Les sorties sont numérotées 0 à 3

- Fonction du démux

  Un démux est un "aiguilleur" :
  - Le démux copie la valeur de e sur la sortie spécifiée par l'adresse
  - le démux met les autres sorties à 0 .

- Équations des sorties : elles sont très simples : voir : **24 Annexe**

- Exemple: démux $1 \rightarrow 4$
  - L'adresse de sortie est codée en 2 bits ($2^2 = 4$) :
    le nombre ba_{bin} permet de sélectionner 1 des 4 sorties S_0, S_1, S_2, S_3
    Par exemple si $ba_{bin} = 10_{bin} = 2_{déc}$ $\Rightarrow S_0 = S_1 = 0, \quad S_2 = e, \quad S_3 = 0$

- Généralisation à un nombre quelconque de bits d'adresses:
  - Pour 2 sorties, il faut 1 bit d'adresse
  - Pour 4 sorties, il faut 2 bits d'adresses
  - Pour 8 sorties, il faut 3 bits d'adresses
  - Pour 16 sorties, il faut 4 bits d'adresses, . . .

- Exemple de circuit : 74HC238

8.4 Le multiplexeur (en abrégé : le "mux")

8.4.1 Définition et fonction

- Entrées / Sortie
  - n entrées "bits d'adresse" notés a, b, c, d, ... (a = poids faible) :
    on y applique un nombre binaire qui spécifie l'adresse d'une des 2^n entrées e_i
  - 2^n entrées, numérotées 0, 1, ... , $2^n - 1$
  - 1 sortie S

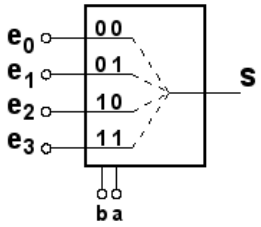


Figure 35 : Mux $4 \rightarrow 1$: diagramme fonctionnel
n = 2 ; il y a 4 entrées numérotées 0 à 3

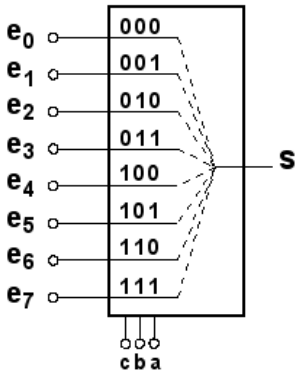


Figure 36 : Mux $8 \rightarrow 1$: diagramme fonctionnel
n = 3 ; il y a 8 entrées numérotées 0 à 7

- Fonction du mux
  - "Multiplexer" c'est sélectionner une des entrées, et copier sa valeur sur la sortie
  - Note : ce multiplexage est appelé **multiplexage temporel** :
    au même endroit (la sortie) apparaissent successivement des infos différentes
  - Un mux fait la fonction inverse d'un démux.

- Équations de la sortie : voir : **24 Annexe**

- Exemple de circuit : 74HC151

  Notez que ce circuit a deux sorties complémentaires.

8.5 Circuits combinant plusieurs mux et/ou démux

8.5.1 Démux à grand nombre d'entrées

- On peut réaliser un démux avec un grand nombre d'entrées,
  en plaçant plusieurs démux en cascade :

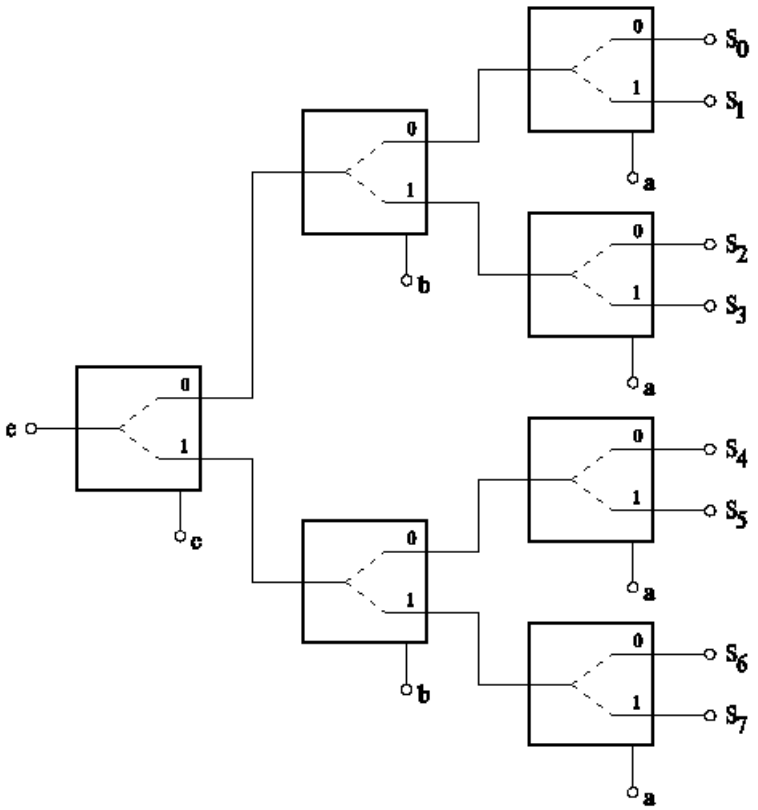


Figure 37 : Démux 1 → 8 réalisé au moyen d'une cascade de démux 1 → 2

- Notez sur le schéma que, pour obtenir l'ordre naturel (0 à 7) des sorties, il faut utiliser
  - la variable de poids fort c dans le démux unique du 1er étage,
  - la variable b dans les 2 démux du 2ème étage.
  - et la variable de poids faible a dans les 4 démux du dernier étage.

8.6 Utilisation d'un Mux comme "Fonction combinatoire universelle"

- Voir **21 Annexe : Réalisation d'une fonction par mux ou démux**

  On dit qu'un mux est une "fonction combinatoire universelle", car il est possible (et facile) d'utiliser un mux pour réaliser n'importe qu'elle fonction logique.

  En pratique, on ne le fait pas car cette solution occupe beaucoup de place sur un circuit imprimé.

8.7 Le Théorème du consensus

8.7.1 Énoncé

$$\overline{s}\,.\,a \;+\; s\,.\,b \;=\; \overline{s}\,.\,a + s\,.\,b + a\,.\,b$$

- Voir exercice : la démonstration de ce théorème est simple : on écrit la table de vérité des deux fonctions $f(s, a, b) = \overline{s}\,.\,a + s.\,b$ et $g(s, a, b) = \overline{s}\,.\,a + s\,.\,b + a\,.\,b$,
  et on constate que les colonnes f et g sont identiques.

8.7.2 Aléa d'une fonction logique

- Considérons un mux 2 → 1, les 2 entrées sont a et b, l'entrée de sélection est s ,
  la sortie du mux est $f = \overline{s}\,.\,a + s\,.\,b$, c.-à-d. que
  - si s = 0 , le mux sélectionne l'entrée a : il copie la valeur de a sur la sortie f
  - si s = 1 , le mux sélectionne l'entrée b : il copie la valeur de b sur la sortie f

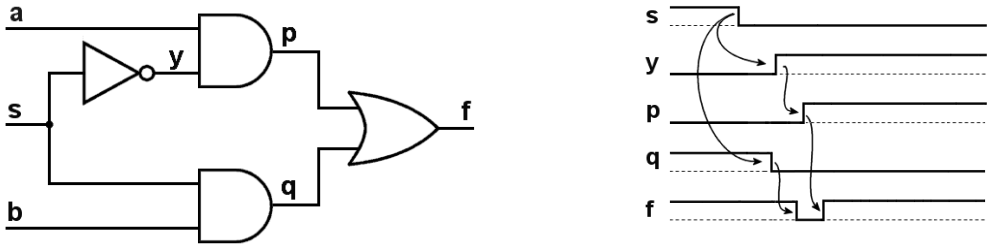


Figure 38 : Circuit mux 2 → 1, et chronogramme (cas a = b = 1, et s passe de 1 à 0)

Dans le cas a = b = 1, on s'attend à toujours observer f = 1 ;
plus précisément, on s'attend à toujours observer f = 1 *quelle que soit la valeur du signal de sélection s : s = 0, s = 1, ou s qui change !*

- Hélas, quand s change, il y a un problème :
  - Commençons dans l'état a = b = s = 1 . L'analyse du schéma montre que y = 0 et donc p = 0 ; d'autre part q = 1 , ⇒ f = 1 : tout est normal jusqu'ici.
  - Faisons alors passer s à 0 (voyez le chronogramme).
    - Si le AND du dessous est rapide, alors q passe à 0 avant que p = 1.
      ⇒ les deux entrées p et q du OR sont à 0 :
      si le OR est rapide, la sortie f passe à 0
    - Après un certain temps, y passe à 1, et ensuite p passe à 1.
      ⇒ la sortie f repasse à 1, et reste ensuite à 1 .

En résumé, quand s change, la sortie f du mux qui valait au départ 1 ,
passe pendant un certain temps (très court : quelques ns) par un état instable 0 ,
avant de revenir à l'état stable 1 !

Ce court passage de la sortie par 0 est appelé "glitch",
et la présence d'un glitch dans le signal de sortie résulte d'un "aléa de fonctionnement" .

Suite à un changement d'une des entrées, la sortie d'un circuit combinatoire qui souffre d'un aléa de fonctionnnement peut afficher (pendant un bref instant) une valeur indésirable instable, temporaire.

8.7.3 Terme de couverture

- Au lieu de réaliser un mux avec $f = \overline{s} \cdot a + s \cdot b$,
  si on ajoute un terme, et qu'on câble $f = \overline{s} \cdot a + s \cdot b + a \cdot b$, on supprime l'aléa :
  Le terme $a \cdot b$ dans la fonction f (et dans le théorème du consensus)
  est appelé "terme de couverture", car c'est sa présence qui supprime l'aléa :

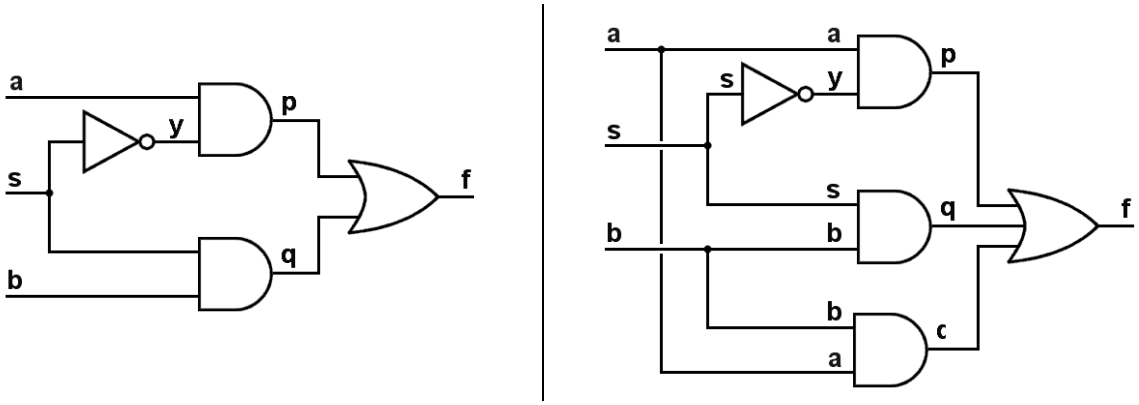


Figure 39 : Circuit mux 2 → 1, à droite complété par un AND de couverture
Si a = b = 1, le signal de couverture c = a . b vaut 1 , et donc : OR(p, q, c) = 1
quelles que soient les valeurs (mêmes transitoires) de p et q lors d'un changement de s .

- L'existence d'aléas est tout-à-fait compatible avec la théorie :
  en effet
  - la logique combinatoire décrit seulement la valeur des sorties d'un circuit combinatoire, une fois que ces sorties sont toutes arrivées dans leur état stable.
  - Le théorème "du consensus" énonce qu'ajouter le terme de couverture $a \cdot b$ à $f = \overline{s} \cdot a + s \cdot b$, c'est-à-dire câbler $f = \overline{s} \cdot a + s \cdot b + a \cdot b$ ne change rien … sous-entendu : *une fois que les sorties sont toutes arrivées dans leur état stable*

- La possibilité d'aléas est implicite dans les datasheets des circuits physiques,
  puisque suite à un changement d'une ou plusieurs des entrées,
  le fabricant spécifie l'état des sorties *après* le délai de propagation du circuit.

- Dans tout circuit combinatoire, on peut ajouter des termes de couverture,
  dans le but de supprimer les aléas = de supprimer des états transitoires non désirés.

8.8 EXERCICES

42. Énoncé : un signal à l'état 1 spécifie qu'un compteur doit compter,
et à l'état 0 ce signal spécifie que le compteur doit décompter.
Choisissez un nom pour ce signal, en vous inspirant des mots anglais Up et Down.

43. Réalisez un décodeur $2 \rightarrow 4$ à sorties actives basses, en utilisant un 74HC138.

44. Le 74HC151 est un mux $8 \rightarrow 1$
Téléchargez, imprimez, lisez et comprenez sa datasheet

45. Réaliser un mux à 8 entrées, à l'aide de multiplexeurs à 4 entrées.
De combien de circuits avez-vous besoin ?

46. Soit une carte multiplexeur à 9 entrées (signaux binaires) et 1 sortie.
    - Combien d'entrées "bits d'adresse" cette carte doit-elle avoir ? (Justifiez)
    - Dessinez le schéma de cette carte, en employant
        - seulement des C.I. (circuits intégrés) du type : "Multiplexeur à 4 entrées"
        - et en employant le moins possible de ces C.I.

47. Démontrez le théorème du consensus au moyen d'une table de vérité

9 Fonctions XOR et XNOR

9.1 Fonction XOR à 2 variables

- La fonction "XOR" = "OU Exclusif" est une fonction
  utilisée notamment dans les circuits de calcul arithmétique (additionneur, soustracteur).

  On note le XOR par le signe "⊕" (le signe "plus" entouré d'un cercle).

- Table de vérité

b	a	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

Tableau 36 : Table de vérité du XOR à 2 variables

  - Exclusif signifie "un seul": la sortie vaut 1 si une et une seule des 2 entrées vaut 1

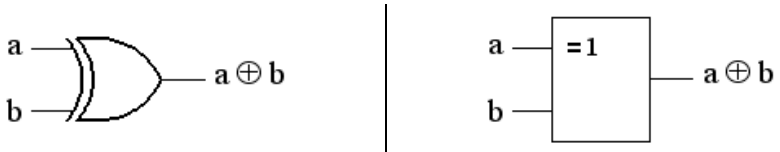
    (en français, XOR correspond à « ou bien »)

    $1 \oplus 1 = 0$

  - Ne confondez pas le XOR (exclusive or), avec le OR (inclusive or) noté "+" :
    (en français, OR correspond à « et/ou »)

    $1 + 1 = 1$

- Logigramme du XOR



Norme MIL **Norme IEC**

Figure 40 : Symboles du XOR (à 2 entrées)

- SDP standard

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

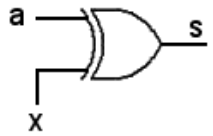
(cette expression n'est pas simplifiable)

9.2 Propriétés du XOR

9.2.1 Inverseur conditionnel

- Ayant $a \oplus 0 = a$ et $a \oplus 1 = \overline{a}$

  $\Rightarrow$ la fonction OU exclusif permet de complémenter ou non une variable a , selon l'état d'une variable de commande x :



si $x = 0$, $s = a$

si $x = 1$, $s = \overline{a}$

Figure 41 : Le OU exclusif est un inverseur conditionnel l'entrée x commande si on inverse ou non

9.2.2 Comparaison de non-égalité (Détecteur de différence)

- Le XOR peut être considéré comme un détecteur de différence. En effet,
  - si les 2 bits sont égaux, la fonction vaut 0 : $a \oplus a = 0$
  - si les 2 bits sont différents la fonction vaut 1 : $a \oplus \overline{a} = 1$

9.2.3 Inversion d'une des entrées d'un XOR

- Si on inverse une des 2 entrées d'un XOR à 2 variables, cela inverse la sortie du XOR
- Démonstration :

Entrées		Sorties			
b	a	$a \oplus b$	$\overline{a}$	$\overline{a} \oplus b$	$\overline{a \oplus b}$
0	0	0	1	1	1
0	1	1	0	0	0
1	0	1	1	0	0
1	1	0	0	1	1
				↑	↑

Tableau 37 : Démonstration : $\overline{a} \oplus b = \overline{a \oplus b}$
(on constate que les 2 colonnes de droite sont identiques)

9.2.4 Propriétés du XOR utiles pour des circuits de calcul arithmétiques

- Commutativité : l'ordre des opérandes n'a pas d'importance :

$$a \oplus b = b \oplus a$$

- Associativité : l'ordre dans lequel on effectue 2 opérations XOR n'a pas d'importance:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

  - Cela a donc un sens de parler du XOR à 3 variables ou plus (voir page suivante)
  - Également, au point de vue notations, on peut omettre les parenthèses:

$$a \oplus b \oplus c = (a \oplus b) \oplus c = a \oplus (b \oplus c)$$

- Distributivité : comme pour le OR inclusif, il y a distributivité du "·" (AND) par rapport au "⊕" (OU exclusif).

$$a \,.\, (b \oplus c) = a \,.\, b \oplus a \,.\, c$$

  Comme en algèbre classique, on peut donc
  - effectuer (passage du membre de gauche au membre de droite), et
  - mettre en évidence (passage du membre de droite au membre de gauche)

- Nous verrons que le XOR est utile pour la construction de circuits arithmétiques (addition, soustraction).

9.3 XOR à 3 entrées

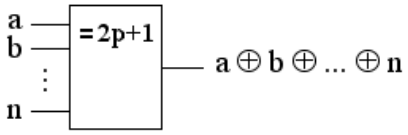
- Table de vérité

c	b	a	a ⊕ b ⊕c
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tableau 38 : XOR à 3 variables : Table de vérité

- "Exclusif" signifie "un seul", mais le nom "OU exclusif" est assez mal choisi :
  - il est vrai qu'à 2 variables il y a bien "exclusivité"
    (la sortie vaut 1 si une et une seule des 2 entrées vaut 1)
  - mais à n variables, il n'y a pas "exclusivité" :
    en fait, à n variables, le XOR vaut 1 si un nombre *impair* d'entrées valent 1
    (il aurait été préférable d'appeler le XOR : "fonction imparité")

- Logigramme IEC du XOR à n entrées (n>2)



XOR à N entrées

Figure 42 : XOR à n entrées (n > 2) – Logigramme IEC

- Fonction "imparité" :
  - à n variables, on ne peut pas utiliser le symbole pour 2 variables :
  - p étant un nombre entier, "2p+1" désigne un nombre impair
    (la sortie vaut 1 si un nombre impair de variables sont à 1)

  Quel que soit le nombre d'entrées :
  - Lorsque toutes les entrées valent "0" la fonction "OU exclusif" vaut "0"
  - Si on complémente 1 entrée (ou un nombre impair d'entrées), la sortie change

  On en déduit :
  - Si on complémente un nombre pair d'entrées, la sortie reste inchangée,
  - Si on complémente 1 entrée (ou un nombre impair d'entrées), la sortie change
  - la sortie vaut "0" s'il y a un nombre pair (0, 2, 4, 6,…) d'entrées à "1"
  - la sortie vaut "1" s'il y a un nombre impair (1, 3, 5,…) d'entrées à "1"

9.4 Bit de Parité et détection d'erreur

- On utilise le XOR dans les circuits de détection et de correction d'erreurs.
  Une application classique est le bit de parité (ou d'imparité) :

- Définition :
  - le "bit de parité" P d'un mot de n bits est la somme modulo-2 des n bits.
    ⇒ le "bit de parité" P est égal au XOR des n bits :
    ⇒ Si les n bits contiennent un nombre pair de 1 , alors P = 0
    Si les n bits contiennent un nombre impair de 1 , alors P = 1 [(1)]
    ⇒ l'ensemble [n bits et P] contient toujours un nombre pair de 1 .

- Exemple :
  - bit de parité d'un mot de 8 bits D_0–D_7 : P = XOR (D0, D1, …, D7)
    ⇒ Si D_0–D_7 contient un nombre pair de 1 , alors P = 0
    Si D_0–D_7 contient un nombre impair de 1, alors P = 1
    ⇒ l'ensemble des signaux [D_0–D_7 et P] contient toujours un nombre pair de 1 .

- Note : puisque $a \oplus b \oplus c = (a \oplus b) \oplus c$,
  on peut réaliser un XOR à n entrées au moyen de plusieurs XOR à 2 entrées :
  Exemple :

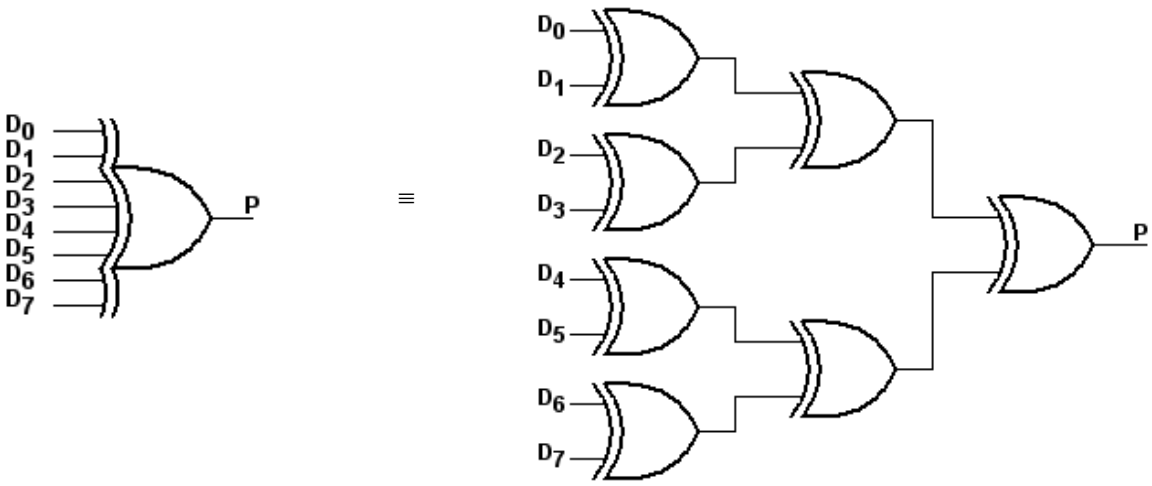


Figure 43 : Bit de parité d'un mot de 8 bits

[(1)] Note : le même mot "parité" designe donc deux concepts différents :

- Ici, la parité ou, plus précisément, le "bit de parité" d'un mot de n bits indique si ce mot contient un nombre impair de 1
- En langage usuel, un nombre entier est "pair" s'il est divisible par 2 (autrement dit, en binaire, si son bit de poids faible vaut 0)

9.4.1 Application : Détection d'une erreur de transmission

- On veut transmettre un mot d'information de 8 bits, $D_0 - D_7$
  On forme le bit de parité P en faisant le XOR des 8 bits de donnée:
  $\Rightarrow$ l'ensemble des 9 signaux $D_0 - D_7$ et P, contient toujours un nombre pair de 1 .
- L'émetteur transmet les 8 bits de donnée + le bit de parité P

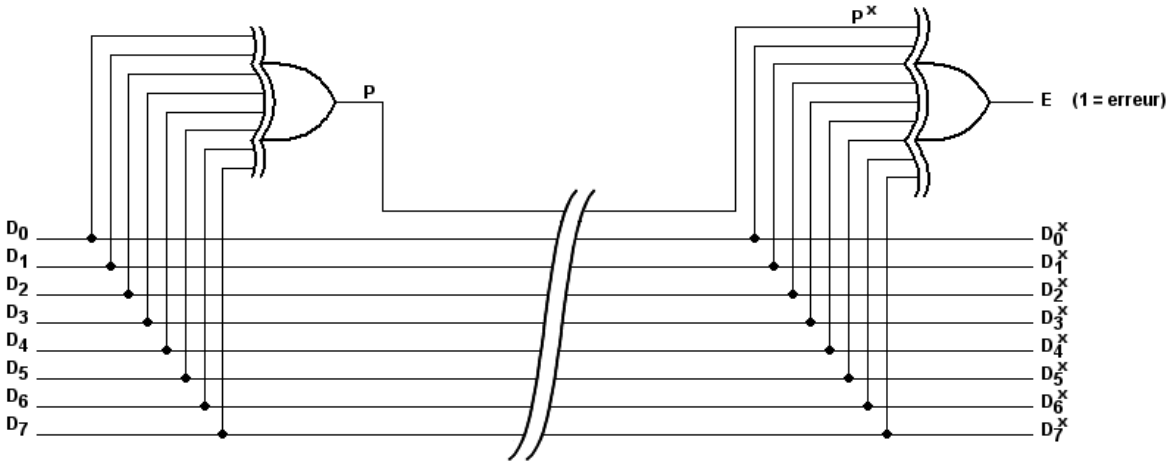


Figure 44 : Détection d'erreur par un bit de parité

- Le récepteur reçoit 9 signaux : D_i^x et P^x. Il en forme le XOR
  - Si la transmission s'est effectuée sans erreur, ce XOR vaut 0 :
    E = 0 indique qu'il y a 0 erreur (ou éventuellement 2, 4, 6 ou 8 erreurs)
  - Si un parasite a changé la valeur d'un seul des 9 bits, la sortie E du XOR vaut 1 .
    E = 1 indique qu'il y a 1 erreur (ou éventuellement 3, 5, 7 ou 9 erreurs)

9.4.2 Efficacité de la détection d'erreur au moyen d'un bit de parité.

- Pour une liaison ou une mémorisation de bonne qualité, peu parasitée,
  - le plus souvent il n'y a pas d'erreur
  - de temps en temps (par ex. pour chaque bit, avec une probabilité de $1/10000$) ,
    il y a une erreur de transmission simple : un bit est reçu avec la mauvaise valeur.
  - Si les erreurs sur 2 bits sont indépendantes,
    la probabilité d'avoir simultanément 2 erreurs est $(1/10000)^2 = 1/100000000$
    $\Rightarrow$ beaucoup d'erreurs sont des erreurs simples : détectables avec un bit de parité
- L'hypothèse « si les erreurs sur 2 bits sont indépendantes » est raisonnable dans certains cas, et l'ajout d'un bit de parité à 8 bits de données est un procédé utile.
  - Il existe des codages plus élaborés que le (seul) bit de parité,
    codages où on ajoute plusieurs bits, ce qui permet de détecter et corriger
    quelques (2 ou 3) erreurs.
  - Ceci dit, un parasite industriel (par ex. causé par le démarrage d'un moteur)
    peut durer longtemps, typiquement quelques millisecondes
    et donc parasiter … plusieurs milliers de bits successifs.
    Certains codes permettent de détecter un grand nombre d'erreurs dans un message (sans pouvoir les corriger: le récepteur demandera une retransmission).

9.5 Fonction XNOR

- Le XNOR est un XOR suivi d'un NOT
- Logigramme

Figure 45 : Symbole du XNOR

- Table de vérité et synthèse

Entrées		XOR
b	a	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Entrées		XNOR
b	a	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1

Tableau 39 : Tables de vérité du XOR et du XNOR à 2 entrées

$\Rightarrow$ SDP standard du XNOR : $\overline{a \oplus b} = \bar{b}.\bar{a} + b.a$

9.5.1 Comparateur (Détecteur d'égalité) de 2 nombres binaires de n bits

- Le XOR est un détecteur de non-égalité
  $\Rightarrow$ le XNOR est un comparateur (détecteur d'égalité) de 2 bits
- Pour vérifier si 2 nombres binaires de n bits sont égaux,
  on vérifie l'égalité de chacune des paires de bits de même poids,
  puis on fait le AND de ces n résultats :

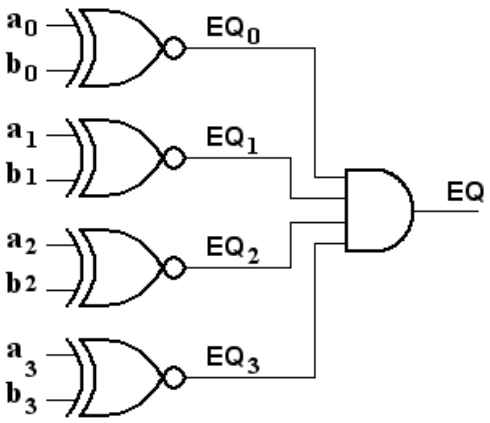


Figure 46 : Comparateur (détecteur d'égalité) à n bits (ici n = 4)

9.6 EXERCICES

48. Donnez les tables de vérité d'une cellule additionnant 2 bits A et B, avec un bit de résultat (S), et un bit de report (R). Dessinez le schéma.

49. …

10 Fonction incomplètement définie

- Dans certains problèmes, la valeur de la fonction n'est pas définie pour certains états d'entrée. Ceci peut survenir pour 2 raisons :
  - les conditions indifférentes (en anglais : "don't care" conditions)
  - les conditions impossibles ("don't happen" conditions)

10.1 Conditions indifférentes ("don't care" conditions)

- Il se peut que pour un ou plusieurs états d'entrée, l'état d'une fonction nous soit égal.
- Exemple: la fonction majorité à 4 entrées donne la valeur de la majorité des entrées.
  Dans les cas où 2 entrées sont à 0 et les 2 autres à 1, peu importe si la sortie vaut 0 ou 1
- À ce stade-ci (définition de la fonction), plutôt que de fixer arbitrairement la valeur de la sortie, il est plus intéressant de laisser provisoirement la fonction non définie
  (on dit aussi : non spécifiée)
- On indique une valeur non définie par une croix "×" ou une barre "–".

10.2 Conditions impossibles ("don't happen" conditions)

- Il se peut que certains états d'entrée ne se présentent jamais ("don't happen" conditions). Plus précisément, certains états ne sont pas censés se présenter en fonctionnement normal.
- Pour ces états d'entrée "impossibles", il est en général économique d'accepter (de spécifer) la valeur « indifférent » (c.-à-d : 0 ou 1, peu importe)

  Exemple : dans un afficheur capable d'afficher la valeur d'un chiffre décimal (de 0 à 9),
  - on doit considérer les états d'entrée de 0 à 9 (0000_{bin} à 1001_{bin}),
  - mais les six états d'entrée $10_{déc}$ à $15_{déc}$ (1010_{bin} à 1111_{bin})
    ne sont pas censés se présenter : peu nous importe ce qui sera affiché
- Pour les états d'entrée impossibles, on indique la valeur indifférente de la fonction également par une croix "×" ou une barre "–" .

10.3 Remarque

- En logique numérique, la sortie ne peut avoir que 2 valeurs : "0" ou "1" ;
  spécifier "×" (ou "–") pour la sortie ne définit *pas* une 3ème valeur de sortie :
  - Lors de la synthèse et de la simplification de la fonction,
    on remplacera chaque valeur de sortie "×" (ou "–"), au choix par "0" ou "1" .
  - Attention :
    Faites ce choix ("0" ou "1") individuellement pour chaque "×" (ou "–") ,
    en choisissant la valeur qui donnera la fonction la plus simple à réaliser .

10.4 Exemple : Fonction majorité

- La fonction majorité est utilisée dans les systèmes "redondants",
  afin d'augmenter la fiabilité en cas de panne : le principe d'un système redondant
  est de réaliser plusieurs systèmes identiques, et de choisir comme résultat
  le résultat obtenu par la majorité des différents systèmes
    - En temps normal, les n systèmes fournissent tous le même résultat,
      et le résultat donné par la fonction majorité est correct
    - En cas de panne, si au moins la moitié des systèmes fournissent un résultat correct, le résultat majoritaire reste correct
    - La fonction majorité à 3 variables d'entrées est complètement définie.
      Il est naturel de laisser incomplètement spécifiée celle à 4 variables d'entrées

10.4.1 Fonction majorité à 3 variables Maj (a, b, c)

n°	c	b	a	Maj (a, b, c)
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Tableau 41 : Fonction majorité à 3 variables
(cette fonction est complètement spécifiée)

- SDP standard : Maj (a, b, c) = $K_3 + K_5 + K_6 + K_7 = \overline{c}ba + c\overline{b}a + cb\overline{a} + cba$
- Forme simplifiée : nous verrons (méthode de Karnaugh) qu'on peut simplifier l'expression ci-dessus : on obtient : Maj (a, b, c) = a b + b c + c a
  En tout cas cette expression est correcte car elle vaut 1 dès qu'au moins 2 variables valent 1

- Logigramme :

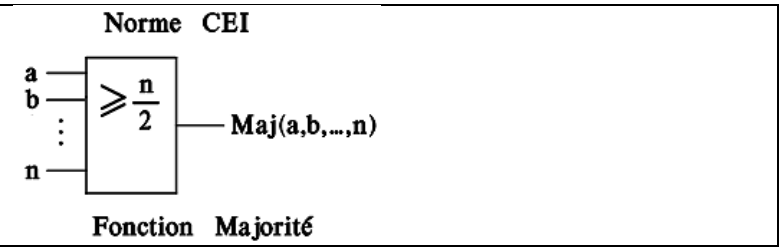


Figure 47 : Logigramme IEC de la fonction Majorité

- L'indication $\geq \frac{n}{2}$ signifie qu'au moins la moitié des variables (la majorité) doivent valoir "1" pour que la sortie vaille "1".
- La fonction majorité n'a pas de symbole MIL.

10.4.2 Fonction majorité à 4 variables Maj (a, b, c, d)

n°	d	c	b	a	Maj (a, b, c, d)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	×
4	0	1	0	0	0
5	0	1	0	1	×
6	0	1	1	0	×
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	×
10	1	0	1	0	×
11	1	0	1	1	1
12	1	1	0	0	×
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Tableau 42 : Fonction majorité à 4 variables : Table de vérité

- Cette table de vérité est un exemple de fonction incomplètement spécifiée :
  - Si les entrées contiennent deux 0 et deux 1,
    on a spécifié la fonction comme indifférente (elle peut au choix valoir 0 ou 1)

10.4.3 Fonction majorité à 5 variables Maj (a, b, c, d, e)

- Cette fonction est complètement spécifiée
- Forme simplifiée :

  On cherche une expression qui vaut 1 dès qu'au moins 3 variables valent 1.

  $\Rightarrow$ Maj (a, b, c, d , e) =

  a b c + a b d + a b e (tous les triplets comprenant a b)

  + a c d + a c e (tous les triplets comprenant a c, mais pas b)

  + a d e (tous les triplets comprenant a d, mais pas b et pas c)

  + b c d + b c e (tous les triplets comprenant b c, mais pas a)

  + b d e (tous les triplets comprenant b d, mais pas a et pas c)

  + c d e (tous les triplets comprenant c d, mais pas a et pas b)

11 Analyse des systèmes combinatoires

11.1 Analyse : définition

- Pour résoudre un problème combinatoire, on fait d'abord l'analyse du problème :
  cela consiste à traduire les "desiderata verbaux" en une table de vérité
  (qui spécifie la sortie désirée pour chaque combinaison des variables d'entrée)
  Faire l'analyse c'est traduire un énoncé (par ex. en français ou anglais) en 0 et 1 !!!
  - Pour certains états d'entrée, la sortie peut être indéfinie
  - S'il y a plusieurs sorties, on doit écrire une table de vérité pour chaque sortie
  - Plus tard, nous procèderons à la synthèse : le passage de la table de vérité à l'équation (ou des tables de vérité aux équations), puis au schéma.
- Afin de se familiariser avec cette démarche de l'analyse, voici un exemple et 2 exercices

11.2 Exemple : Décodeur hexadécimal-7segments

- Cahier de charge
  - Un "afficheur 7 segments" se présente comme suit :

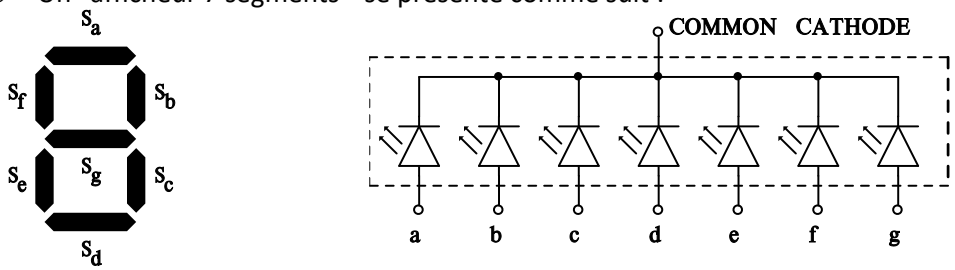


Figure 48 : Afficheur "à cathode commune"

Chacun des 7 segments est une LED (Light Emitting Diode, diode électroluminescente). On connecte le cathode commune au GND (0V) ;
si on applique une tension suffisamment élevée
sur une des entrées, la LED s'allume [(1)]. Sinon elle est éteinte

- Sur les entrées d c b a d'un décodeur pour afficheur 7 segments,
  on applique un nombre compris entre 0_{hex} et F_{hex}.
  Le décodeur doit piloter l'affichage 7 segments, et lui faire allumer
  les segments ci-dessous :

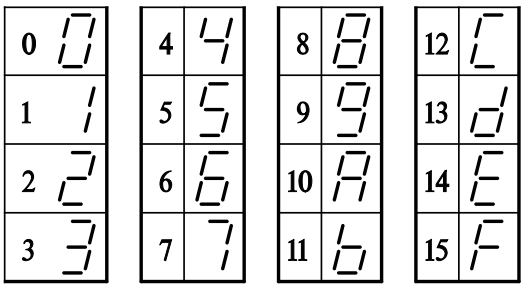


Figure 49 : Segments à allumer sur un afficheur 7 segments en fonction du nombre appliqué à l'entrée du décodeur

[(1)] Voir cours d'électronique analogique. Attention, il faut limiter le courant de chaque LED, par exemple par une résistance, car sinon la surintensité brûle instantanément la LED.

- Analyse

  - Il y a 4 variables d'entrée (qui donnent en binaire le chiffre hex à afficher)
    Il y a 7 sorties, donnant les valeurs associées aux 7 segments de l'afficheur.
  - Appelons d, c, b, a les 4 variables. Convenons de donner le poids fort à d, le faible à a.
    Appelons S_a, S_b, S_c, S_d, S_e, S_f, S_g les 7 fonctions associées aux 7 segments.
    L'afficheur étant à cathode commune, $S_j = 1$ allume le segment j [(1)]
  - Représentons par un bloc rectangulaire le circuit matérialisant le 7 fonctions réalisant cette table de vérité; ce circuit aura 4 entrées d c b a et 7 sorties Sa →Sg :

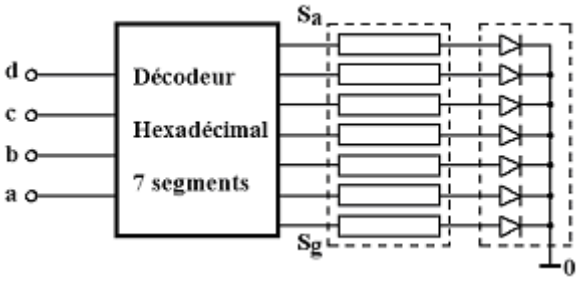


Figure 50 : Décodeur hex vers 7 segments, et afficheur 7 segments à cathode commune
On a dessiné les résistances de limitation du courant

  - D'où la table de vérité donnant les 7 fonctions S_a, ..., S_g en fonction des variables dcba :

n°	d	c	b	a	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	1	1	1	0	1	1	1
11	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	0	1	0	0	1	1	1	0
13	1	1	0	1	0	1	1	1	1	0	1
14	1	1	1	0	1	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	1	1	1

Tableau 43 : Tables de vérité des 7 fonctions d'un afficheur 7 segments

  - Remplir cette table termine la phase d'analyse du problème.
    Nous verrons plus loin comment matérialiser cette table de façon économique,
    ce qu'on appelle "faire la synthèse" (ici de 7 fonctions).

[(1)] Un afficheur à anode commune nécessiterait le choix inverse : un 0 allumerait le segment.

11.3 EXERCICES

50. Affichage d'un "dé électronique"

- Énoncé :

  On souhaite visualiser 6 états différents, numérotés de 1 à 6 et codés à l'aide de 3 bits c b a qui prennent les états cba = 001 jusque cba = 110, sur un affichage imitant la surface d'un dé : chacun des 7 points est remplacé par une LED :

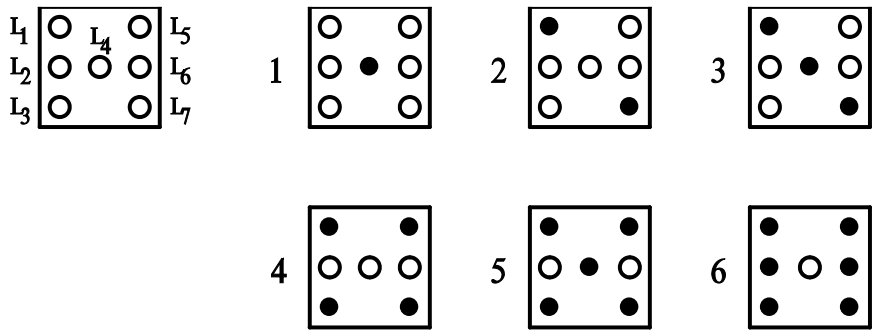


Figure 51 : Exercice : Dé électronique

Le dessin à gauche donne la disposition et la numérotation des LEDS L_1 à L_7.

Les 6 autres dessins donnent la disposition adoptée pour les 6 chiffres 1 à 6
(un cercle vide indique une LED éteinte, un cercle plein une LED allumée).

On demande la table des fonctions de commande des 7 LEDs en function des c b a.

51. Distributeur de boisson élémentaire

- <u>Énoncé</u> :
  - on désire réaliser un distributeur de boisson offrant 3 possibilités :
    - de l'eau
    - du cassis à l'eau
    - de la menthe à l'eau
  - Les organes de commande du panneau avant du distributeur sont 3 boutons :
    - e : commande de l'eau
    - m : commande de la menthe
    - c : commande du cassis

    L'utilisation prévue est la suivante : pour demander
    - de l'eau : appuyer sur e
    - de la menthe à l'eau : appuyer sur e et sur m
    - du cassis à l'eau : appuyer sur e et sur c

    mais on ne pourra jamais obtenir les combinaisons suivantes :
    - de la menthe seule
    - du cassis seul
    - de la menthe en même temps que du cassis
  - On demande la table de définition de la fonction R
    qui commande le relais ouvrant la vanne délivrant l'eau
    On respectera la convention : R = 1 ouvre le robinet

12 Synthèse des systèmes combinatoires. Intro. : Table de Karnaugh

12.1 Synthèse. Simplification. Somme minimale. Méthodes de simplification.

- La synthèse consiste à traduire la table de vérité en équations (et à en déduire le schéma)

- La SDP standard directement déduite de la table de vérité,
  donne un schéma dont la *structure* est simple (des NOTs, des ANDs, et un OR),
  mais qui utilise en général beaucoup de composants :
  il existe souvent des schémas équivalents[(1)] plus simples.

- "Transformer" une expression algébrique c'est
  passer d'une expression à une autre expression, équivalente.
  Un cas particulier de transformation est la simplification :

- Simplifier un polynôme, c'est le transformer en un autre polynôme équivalent,
  qu'on appelle "SDP minimale" (Somme De Produits minimale)

  Définition :

- Une SDP "minimale"
  - comporte le minimum de termes, et
  - chaque terme contient le minimum de facteurs

- On obtiendra ainsi un schéma simple. Physiquement, le circuit peut être réalisé en
  - NOT-AND-OR (traduction immédiate de la SDP minimale),
    ou en NAND (par utilisation de De Morgan),
  - ou en circuits programmables.

- Il existe plusieurs méthodes de synthèse/simplification, dont :
  - la simplification algébrique libre. Elle est quasi inutilisable par l'être humain.
  - la méthode de Karnaugh[(2)]. C'est celle que nous utiliserons.
    Cette méthode intuitive convient bien à l'être humain
    pour des fonctions de petites tailles, jusque 4 ou 5 variables
  - la méthode de Mac Cluskey. Elle est peu adaptée à l'être humain,
    car elle est très systématique mais très lourde. Nous ne l'étudierons pas.
    Cette méthode est implémentée sous forme de programmes informatiques.

- Note : quand on conçoit un circuit, en plus de la simplicité, on doit considérer d'autres critères, par exemple : le prix des circuits, leur disponibilité (stock, délais d'approvisionnement), la facilité de compréhension et donc la facilité de maintenance.

  Ces critères ne sont <u>pas</u> pris en considération par les méthodes de simplification.

[(1)] Définition : Des expressions algébriques et des circuits sont dits équivalents" ou "égaux"
s'ils réalisent la même fonction, c.-à-d. s'ils ont la même table de vérité.

[(2)] Publiée en 1950 par Maurice Karnaugh, aux laboratoires Bell.

12.2 Rappel : Disposition d'une Table de vérité

N° d'état	Entrées				Sortie
	d	c	b	a	
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	…

Tableau 46 : Disposition d'une Table de vérité de 4 variables
Ici n = 4 : il y a $2^4 = 16$ états d'entrée = 16 lignes

- Pour n variables d'entrée, la Table de vérité contient 2^n lignes
- Chaque ligne correspond à 1 état d'entrée :
  - Les lignes sont numérotées dans l'ordre naturel $0, 1, 2, \ldots, 2^n - 1$
  - Dans la case "sortie" de chaque ligne, on écrit la valeur désirée de la fonction

12.3 Table de Karnaugh (= Diagramme de Karnaugh = Karnaugh-map = K-map)

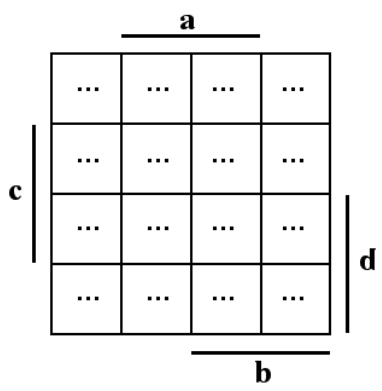
- Une Table de Karnaugh est une *copie des 2^n cases " Sorties " de la Table de vérité, dans un ordre particulier : les n^{os} de 2 cases adjacentes[(1)] diffèrent par un seul bit.*

  Vocabulaire : dans une table de Karnaugh, 2 cases adjacentes sont dites "connexes", ou "contiguës" (c.-à-d. que leur numéro diffèrent pas un seul bit) .

12.3.1 Table de Karnaugh à 4 variables

- Une Table de Karnaugh de 4 variables contient $2^4 = 16$ cases :

(1) Définition : deux cases qui ont un côté en commun sont dites "adjacentes" (on dit aussi "voisines") .

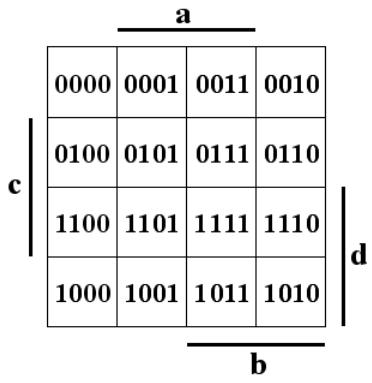


ba / dc	00	01	11	10
00	…	…	…	…
01	…	…	…	…
11	…	…	…	…
10	…	…	…	…

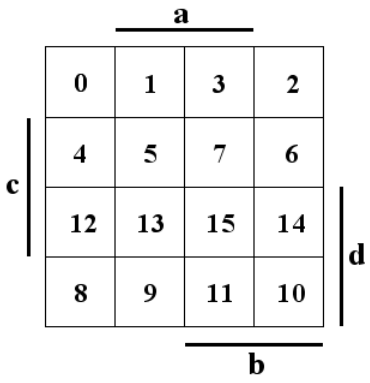
Figure 52 : Table de Karnaugh à 4 variables
Ces 2 dessins sont équivalents : utilisez celui que préférez.

- La barre $\overline{a}$ indique les colonnes où a=1 . Dans les autres colonnes, a=0
- La barre $\overline{b}$ indique les colonnes où b=1 . Dans les autres colonnes, b=0
- La barre c indique les lignes où c = 1 . Dans les autres lignes, c = 0
- La barre d indique que les lignes où d = 1 . Dans les autres lignes, d = 0

- Normalement, on écrit la valeur de la fonction de sortie, dans chaque case de la table. Mais pour montrer clairement l'ordre des cases déduit de la **Figure 52**, écrivons ici son numéro dans chacune des cases :



Dans chaque case on a écrit l'état d'entrée dcba



Idem, converti en décimal

Figure 53 : Numérotation des cases d'une Table de Karnaugh à 4 variables

12.3.2 Ordre des cases et remplissage de la Table de Karnaugh depuis la Table de Vérité

- L'ordre des cases de la Table de Karnaugh est assez facile à retenir en décimal :
  - Dans une ligne on écrit les numéros décimaux des cases de gauche à droite, sauf qu'on intervertit l'ordre pour les 2 dernières colonnes
  - On écrit les lignes de haut en bas, mais on intervertit les 2 dernières lignes
- On lit les cases de la colonne "Sortie" de la Table de vérité de haut en bas [(1)] , et on les copie une à une dans la table de Karnaugh, *dans l'ordre des cases de celle-ci.*

[(1)] Ceci suppose bien entendu qu'on a écrit la Table de vérité dans l'ordre naturel des états d'entrée.

12.3.3 Table de Karnaugh à 3 variables

- Rappel : disposition d'une Table de vérité, ici à n = 3 variables :

N° d'état	Entrées			Sortie
	c	b	a	
0	0	0	0	…
1	0	0	1	…
2	0	1	0	…
3	0	1	1	…
4	1	0	0	…
5	1	0	1	…
6	1	1	0	…
7	1	1	1	…

Tableau 47 : Disposition d'une Table de vérité de 3 variables (n = 3, 8 lignes)

- Une Table de Karnaugh de 3 variables contient $2^3 = 8$ cases :

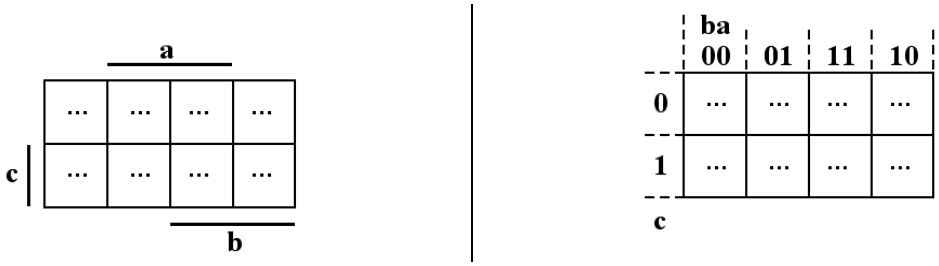
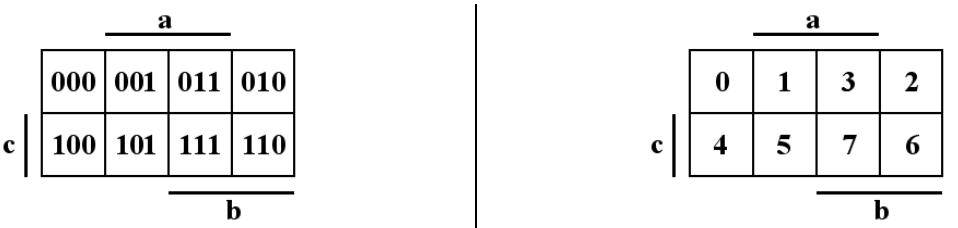


Figure 54 : Table de Karnaugh à 3 variables
Ces 2 dessins sont absolument équivalents : utilisez celui que préférez.

- Normalement, on écrit la valeur de la fonction de sortie, dans chaque case de la table.
  Mais pour montrer clairement l'ordre des cases déduit de la **Figure 54,**
  écrivons ici son numéro dans chacune des cases :



Dans chaque case, on a écrit l'état d'entrée cba

Idem, converti en décimal

Figure 55 : Numérotation des cases d'une Table de Karnaugh à 3 variables

- Remarque : la table à 3 variables est la moitié supérieure de la table à 4 variables
  (cela revient à ne considérer que les cases d = 0 dans la table à 4 variables)

12.3.4 Table de Karnaugh à 2 variables

- Rappel : disposition d'une Table de vérité, ici à n = 2 variables :

N° d'état	Entrées			Sortie
	c	b	a	
0	0	0	0	…
1	0	0	1	…
2	0	1	0	…
3	0	1	1	…

Tableau 48 : Disposition d'une Table de vérité de 2 variables (n = 2, 4 lignes)

- Une Table de Karnaugh de 2 variables contient $2^2 = 4$ cases :

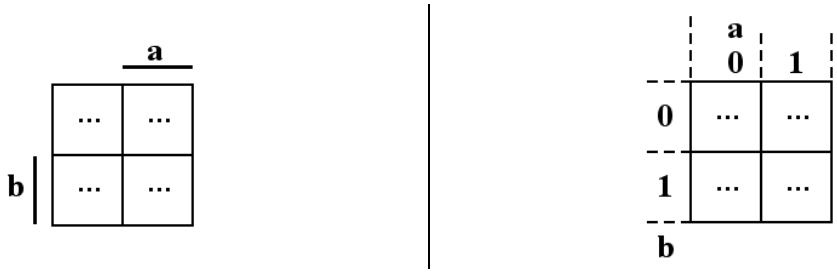
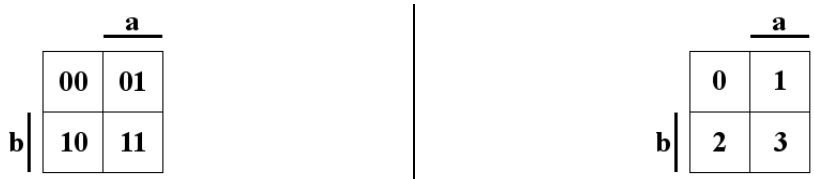


Figure 56 : Table de Karnaugh à 2 variables (ces 2 dessins sont équivalents)

- Normalement, on écrit la valeur de la fonction de sortie, dans chaque case de la table.
  Mais pour montrer clairement l'ordre des cases déduit de la **Figure 56**,
  écrivons ici son numéro dans chacune des cases :



Dans chaque case, on a écrit l'état d'entrée (les bits b a)

Dans chaque case on a écrit le n° d'état d'entrées b a , converti en décimal

Figure 57 : Numérotation des cases d'une Table de Karnaugh à 2 variables

- Remarques :
  - On peut définir de même une Table de Karnaugh à 1 variable
    (cette table aura 2 cases), mais c'est peu utile
  - On peut définir des tables de Karnaugh à 5 variables ou plus (voir : **20 Annexe**),
    mais elles deviennent difficiles à utiliser en pratique.

13 Synthèse : Méthode de Karnaugh

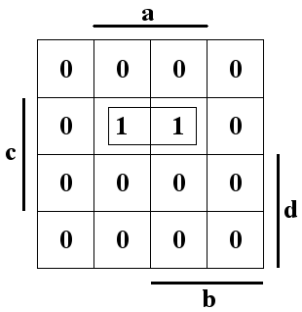
- Les explications fournies dans ce chapitre concernent une fonction de 4 variables

13.1 Minterms et cases de la table de Karnaugh

- Rappel : un minterm est un produit de toutes les variables d'entrée (complémentées ou non : pour n variables, il y a 2^n minterms)
- Chaque case de la table de Karnaugh correspond à un minterm particulier

13.2 Regroupements de 2 cases contiguës en un pavé

- On peut regrouper 2 cases adjacentes en un seul "pavé" si ces cases sont à "1" ou "×" (don't care), c.-à-d. si aucune des cases ne contient "0"
- Le pavé obtenu correspond à un monôme qui aura perdu une variable par rapport aux 2 cases qu'on a regroupé.



Exemple :

la SDP standard de F est une somme de 2 minterms à 4 lettres :

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

Regroupons les 2 cases adjacentes qui contiennent un "1" :

ce rectangle correspond à un monôme de 4 – 1 = 3 variables

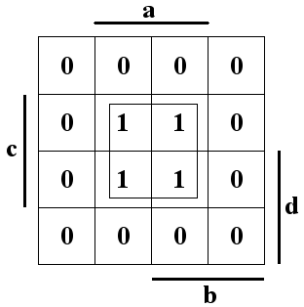
$F = \bar{d}.c.\bar{b}.a + \bar{d}.c.b.a = \bar{d}.c.a.(\bar{b} + b) = \bar{d}.c.a$

Figure 58 : Table de Karnaugh, groupement de 2 cases adjacentes

13.3 Regroupements de 2 pavés contigus en un seul pavé

- On peut regrouper 2 pavés contigus (chacun de 2 cases), pour en faire un pavé de 4 cases si toutes ces cases sont à "1" ou à "×" (autrement dit si aucune des cases ne contient "0")
- Le pavé obtenu correspond à un monôme qui aura perdu une variable par rapport aux 2 pavés qu'on a regroupé.

Exemple : dans le tableau de Karnaugh ci-dessous, on peut regrouper 4 cases adjacentes :



Un "pavé" est

- un regroupement de cases adjacentes
- qui contiennent des "1" (ou des "×")
- qui forment un rectangle (ou un carré),
- la longueur de chaque côté d'un pavé est une puissance de 2 : 1, 2 ou 4

Figure 59 : Table de Karnaugh, groupement de 4 cases adjacentes en un rectangle

13.4 Trouver le monôme correspondant à un pavé

- Pour trouver l'expression du monôme correspondant à un pavé :
  on regarde successivement ce que vaut chaque variable, dans toutes les cases du pavé :
  - Si une variable est à "**1**" dans toutes les cases du pavé :
    on écrit cette variable **sous forme vraie** dans le monôme.
  - Si une variable est à "**0**" dans toutes les cases du pavé :
    on écrit cette variable **sous forme complémentée** dans le monôme.
  - Si une variable est à "1" dans certaines cases du pavé, et à "0" dans d'autres cases du pavé, on n'écrit pas cette variable dans le monôme.
  - A chaque regroupement, une variable disparaît : au plus le pavé obtenu est grand, au plus le monôme est simple, contient moins de variables.
- Dans l'exemple **Figure 59** page précédente :
  - Les 4 cases à "1" sont sous la barre $\underline{\mathbf{a}}$ $\Rightarrow$ a apparaîtra sous forme vraie
  - Certaines (3ème colonne) de ces 4 cases sont au-dessus de la barre $\overline{\mathbf{b}}$ et d'autres (2ème colonne) sont en dehors $\Rightarrow$ b n'apparaît pas.
  - Par un même raisonnement : c apparaît sous forme vraie, et d n'apparaît pas.

  Bref, le pavé dessiné correspond au monôme c . a (monôme de 2 variables)

13.5 Continuation des regroupements, et monômes correspondants

- Au vu des "1" et "×" dans la table, si c'est possible on regroupe 8 cases adjacentes formant un rectangle. Ce rectangle correspondra à un monôme d'une seule variable.
- Toujours à 4 variables, un pavé de 16 cases correspond à une fonction qui vaut "1" ou "×" dans toutes les cases : il s'agit de la constante logique "1" .
- Plus le pavé est grand, plus l'expression du monôme associé est simple.
  $\Rightarrow$ Exercez-vous à repérer dans la table les pavés les plus grands possibles,
  qui ne sont pas complètement inclus dans un autre.

13.6 Les colonnes et les lignes extérieures de la Table de Karnaugh sont connexes

- ATTENTION : Considérez la table de Karnaugh comme un tore :
  les colonnes extérieures sont connexes, et les lignes extérieures sont connexes !!!.

  Par exemple, les 4 cases à 1 de la table **Figure 60** forment bien un carré (de côté 2) !

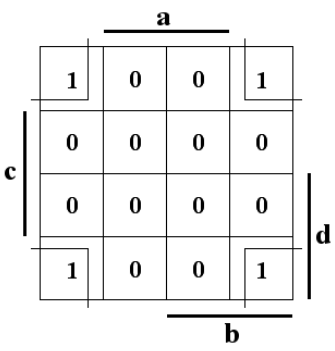


Figure 60 : Une table de K est un tore

- On indique un tel pavé en entourant les 4 cases, et en prolongeant les lignes vers l'extérieur pour suggérer qu'elles rejoignent les cases adjacentes de l'autre côté.
- La case en haut à gauche est connexe à celle en haut à droite : seule la variable b change entre ces 2 cases (à gauche b = 0, à droite b = 1) et les variables d c a ont le même état 0 0 0 .
- La case en haut à gauche est connexe à celle en bas à gauche : seule la variable d change (en haut d = 0, en bas d = 1), et les variables c b a ont le même état 0 0 0 .
- Le monôme associé à ce pavé de 4 cases est : $\bar{c} . \bar{a}$

13.7 Cas des fonctions incomplètement spécifiées

- Note : les cases marquées "×" *peuvent* mais ne *doivent* pas être englobées dans des pavés :
  - elles peuvent l'être (et c'est utile si cela permet d'obtenir des pavés plus grands)
  - mais on ne les englobe pas dans un pavé si cela nécessite un pavé supplémentaire.
- Dans le résultat obtenu, dans la fonction réalisée:
  - les cases "×" englobées dans (au moins) un pavé seront en fait à "1"
  - les cases "×" qui ne sont englobées dans aucun pavé seront en fait à "0" .

13.8 Résumé de la méthode de simplification par table de Karnaugh

- Balayer la table de Karnaugh (par exemple de gauche à droite, et de haut en bas),
- Tant qu'il reste dans la table au moins une case à "1" non incluse dans un pavé :
  - créer un pavé couvrant cette case ; faire "grandir" ce pavé le plus possible, mais en regroupant seulement des cases adjacentes qui contiennent "1" ou "×"
  - (ce pavé est donc un rectangle ou un carré, le plus grand possible ; la longueur de ses côtés doit être de 1 , 2 ou 4 cases).
- Résultat final : la fonction est la SDP minimale, c'est la somme des monômes correspondants aux différents pavés dessinés

13.9 Synthèse du complément de la fonction

- Si une fonction f contient beaucoup de cases à 1 et peu de cases à 0 , on conseille ceci :
  - Synthétiser la fonction f ,
  - Synthétiser également la fonction $g = \bar{f}$:
    - On obtient la table de vérité de g en copiant celle de f mais en remplaçant les 0 par des 1, et les 1 par des 0
    - les "×" restent des "×"

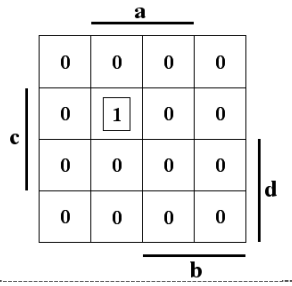
    en effet il est possible que g ait une expression simple
  - Ayant synthétisé g , un NOT suffit pour obtenir la fonction demandée : $f = \bar{g}$
  - On peut alors examiner les deux circuits, et choisir le plus simple.

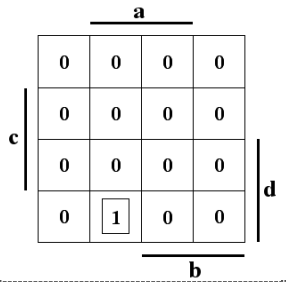
13.10 Pour information : simplification supplémentaire grâce au XOR

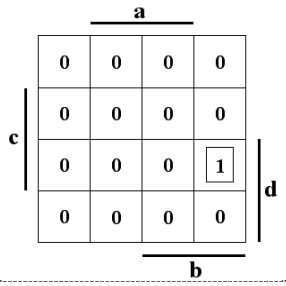
- Dans une table de Karnaugh, on trouve parfois des cases à "1" réparties en damier : dans ce cas, on utilise parfois des portes XOR.
  Sauf pour les circuits comparateurs d'égalité et les additionneurs, on utilise peu cette méthode car elle complique la compréhension du circuit.

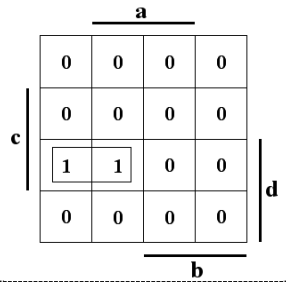
13.11 EXERCICES

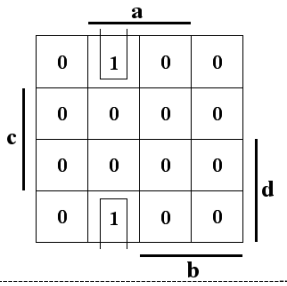
52. Identifiez les monômes correspondant aux pavés dessinés ci-dessous :

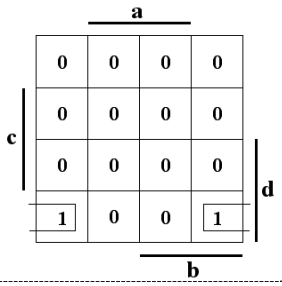


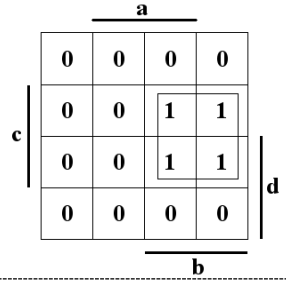


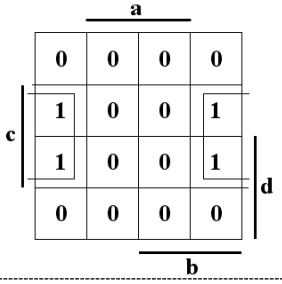


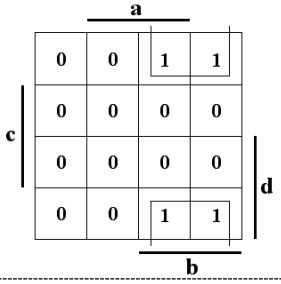


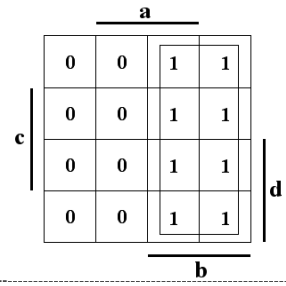


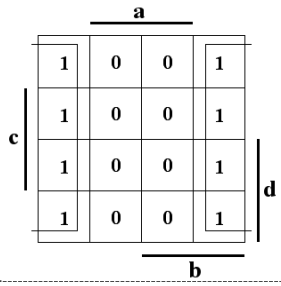


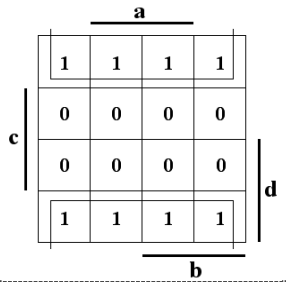


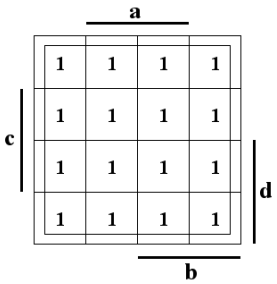


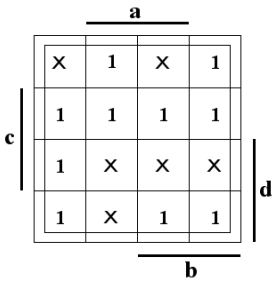












53. Énoncé : Faites la synthèse de la fonction F par la méthode de Karnaugh

Entrées					Sortie
n°	d	c	b	a	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	x
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	x
13	1	1	0	1	x
14	1	1	1	0	0
15	1	1	1	1	1

Tableau 49 : Exercice : Table de vérité d'une fonction F à synthétiser

Solution :

54. Faites une synthèse minimale de la fonction F donnée ci-dessous.
Même question pour la fonction Z

N	d	c	b	A	F	Z
0	0	0	0	0	1	×
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	0	1	1	0	1
4	0	1	0	0	1	1
5	0	1	0	1	1	0
6	0	1	1	0	1	0
7	0	1	1	1	1	0
8	1	0	0	0	0	1
9	1	0	0	1	×	1
10	1	0	1	0	1	×
11	1	0	1	1	×	0
12	1	1	0	0	0	1
13	1	1	0	1	1	1
14	1	1	1	0	0	0
15	1	1	1	1	1	0

(" × " signifie "indifférent")

Tableau 50 : Deux fonctions à synthétiser

Solution :

:

14 Opérations arithmétiques sur les entiers naturels exprimés en binaire

14.1 Tables des 4 opérations arithmétiques sur des nombres de 1 chiffre

- Vous avez appris par cœur les tables des 4 opérations (+ , − , × , /)
  sur des entiers naturels d'un seul chiffre décimal :

+	0	1	2	…	9
0	0	1	2	…	9
1	1	2	3	…	10
2	2	3	4	…	11
…	…	…	…	…	…
9	9	10	11	…	18

×	0	1	2	…	9
0	0	0	0	…	0
1	0	1	2	…	9
2	0	2	4	…	18
…	…	…	…	…	…
9	0	9	18	…	81

−	0	1	2	…	9
0	0	9	8	…	1
1	1	0	9	…	2
2	2	1	0	…	3
…	…	…	…	…	…
9	9	8	7	…	0

Tableau 51 : Extraits de Tables des opérations pour des nombres décimaux de 1 chiffre

- Les tables des 4 opérations sur des entiers naturels d'un seul chiffre binaire sont :

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

−	0	1
0	0	1
1	1	0

:	0	1
0	×	0
1	×	1

Tableau 52 : Table des 4 opérations pour des nombres binaires de 1 chiffre

- Réécrivons ces tables en notant séparément les résultats, les reports et les emprunts :

Addition	Multiplication
0 + 0 = 0	0 × 0 = 0
0 + 1 = 1	0 × 1 = 0
1 + 0 = 1	1 × 0 = 0
1 + 1 = 0, report = 1	1 × 1 = 1

Soustraction	Division
0 − 0 = 0	0 : 0 = × (indéterminé)
0 − 1 = 1, emprunt = 1	0 : 1 = 0
1 − 0 = 1	1 : 0 = × (impossible)
1 − 1 = 0	1 : 1 = 1

Tableau 53 : Tables des 4 opérations pour des nombres binaires de 1 chiffre

14.2 Opérations arithmétiques sur des nombres de plusieurs chiffres

- Pour effectuer les 4 opérations sur des nombres de plusieurs chiffres,
  les méthodes usuelles (calculs en colonnes successives, avec reports et emprunts)
  sont applicables aux nombres écrits au moyen d'un système de numération de position, quelle que soit la base : base 10, base 16, base 2, ou toute autre base !
- Exemple : addition de nombres binaires : $1101_2 + 1001_2$: On écrit :

```
reports      1  1
              1101
        +     1001
        =    10110      ⇒ c'est bien le même résultat qu'en décimal : 13 + 9 = 22déc
```

Nous verrons plus loin le circuit binaire "additionneur", qui réalise cette fonction.

14.3 EXERCICES

55. Réalisez les calculs suivants en binaire, et vérifiez votre réponse en base 10 :

    - $100111_2 + 010010_2$

    - $100111_2 - 010010_2 =$

    - $10010_2 \times 101_2 =$

    - $10010_2 \ / \ 101_2 =$

15 Systèmes itératifs

15.1 Introduction

- La synthèse par Karnaugh n'est efficace que pour un petit nombre de variables.
  Pour réaliser un système plus complexe, on décompose le problème en plusieurs sous-problèmes plus simples.
- Parfois, ces sous-problèmes sont *exactement identiques* :
  il suffit alors d'en résoudre un seul, et de câbler un système itératif,
  c.-à-d. composé de n *sous-ensembles exactement identiques*, *connectés "en cascade".*

  Ces systèmes itératifs permettent par ex. de réaliser les additionneurs, soustracteurs, comparateurs, et d'autres fonctions numériques usuelles.
  Exemple: l'addition binaire est réalisable bit par bit par un circuit itératif.

15.2 Définition et structure générale d'un système itératif

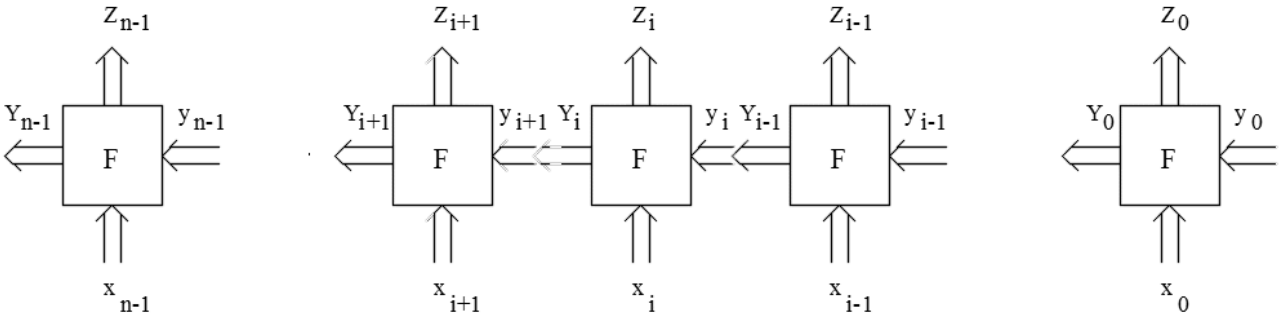


Figure 67 : Système itératif

Un système itératif est un système constitué de n sous-systèmes qui réalisent exactement les mêmes fonctions, mais à partir de variables d'entrée différentes (dessin ci-dessus) :

- Les variables d'entrée x_i sont les variables externes des systèmes combinatoires.
- Les variables y_i sont des variables internes appelées variables interstitielles.
- Chaque bloc réalise individuellement les mêmes fonctions de sortie Z_i externes et Y_i internes. Un bloc est connecté uniquement à ses 2 blocs voisins immédiats : les sorties interstitielles Y_{i-1} sont connectées aux entrées interstitielles y_i de la cellule de rang immédiatement supérieur:

$$y_i = Y_{i-1}$$

15.3 Avantages et inconvénients des systèmes itératifs

- Avantages : simplicité et possibilité d'extension: mettre "en cascade" deux ou plusieurs systèmes permet de créer un système avec autant de bits que désiré.
- Inconvénient : lenteur : après modification des entrées, le délai de stabilisation de la sortie de poids le plus haut augmente proportionnellement au nombre n d'étages !

  Par exemple, si une cellule occasionne un délai égal à 2 délais de portes
  (cas d'une synthèse AND-OR ou NAND-NAND),
  il faut $n \times 2$ délais de porte pour que la dernière sortie soit stabilisée.

15.4 L'additionneur binaire arithmétique (c.-à-d. pour des nombres ≥ 0)

- L'additionneur est un circuit très utilisé, et c'est l'exemple type de circuit itératif.

15.4.1 L'addition décimale

- Soit à calculer S = A + B ; on considère seulement le cas des nombres positifs: $A \geq 0$, $B \geq 0$ (nous traiterons plus loin les nombres négatifs).
- L'algorithme (= succession des opérations, procédé) d'addition en décimal est celui-ci :
  - On fait le calcul en partant du rang de poids le plus faible
  - Dans un rang, si le résultat de l'addition dépasse 9 , on fait un "report" vers le rang suivant, c.-à-d. :
    - on soustrait 10 (dix) unités au résultat du rang traité (autrement dit, on réalise l'addition modulo dix),
    - et on ajoute une unité au rang suivant

Par exemple, calculons S = 28 + 44 :

Pondération	10^2	10^1	10^0
Retenue	0	1	(0)
A		**2**	**8**
B		**4**	**4**
S		**7**	**2**

Tableau 54 : Addition de 2 nombre de 2 chiffres décimaux

  - Rang des unités : 8 + 4 = 12 est > 9 $\Rightarrow$ on fait un report :
    - Au résultat de ce rang, on soustrait 10 : on calcule donc 8 + 4 – 10 = 2
  - Rang des dizaines : il y a un report de 1 $\Rightarrow$ on calcule 1 + 2 + 4 = 7
  - Il n'y a pas de report ni de chiffres des centaines : on a fini. Le résultat est S = $72_{déc}$

15.4.2 Addition binaire arithmétique (= pour des entiers naturels = positifs ou nuls)

- Le procédé en binaire est identique :
  - On fait le calcul en partant du poids le plus faible, et on progresse vers la gauche
  - dans un rang, si l'addition a un résultat > 1, on fait un report vers le rang suivant
    - on soustrait 2 unités au résultat du rang traité : 1 +1 – 2 = 0 (autrement dit, on réalise une addition modulo 2)
    - on ajoute une unité au rang suivant

Exemple : A + B où A = $28_{déc} = 1C_{hex} = 00011100_{bin}$ et B = $44_{déc} = 2C_{hex} = 00101100_{bin}$

Pondération	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
retenue	0	0	1	1	1	1	0	0	(0)
A		**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
B		**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
S		**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**

= 48_{hex} = $72_{déc}$

Tableau 55 : Addition de 2 nombres de 8 bits

Ex.: bits de rang 2^3 : 1 (= report) + 1 + 1 = 3 > 1 ; en binaire, 3 est représenté par 11, et donc :

- le bit de résultat pour le rang 2^3 est à 1
- le bit de report est à 1 pour le rang suivant 2^4

15.4.3 Cellule "demi-additionneur" = additionneur sans report à l'entrée = "Half-Adder"

- Vocabulaire:
  - LSB = Least Significant Bit = le bit de poids le plus faible
  - MSB = Most Significant Bit = le bit de poids le plus fort

- Dans une addition binaire, le calcul du LSB du résultat est simple, car il n'y a pas d'entrée report

  Un additionneur qui n'a pas d'entrée report est appelé : "demi-additionneur", en anglais "Half-Adder" (nous allons voir pourquoi aux pages suivantes).

  Le Half-Adder a une sortie S (Somme), il n'a pas d'entrée Cin (report), il a une sortie Cout (report).

Entrées		Sorties	
b	a	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tableau 56 : Half-Adder : Tables de vérité

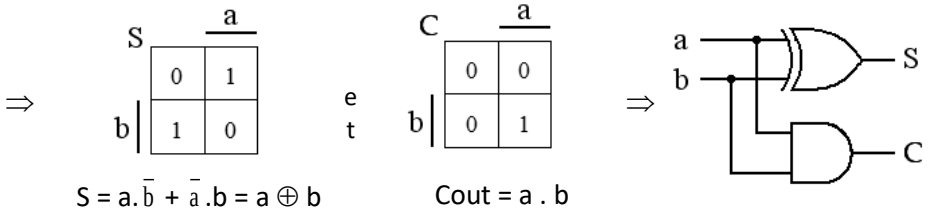


Figure 68 : Half-Adder : Tables de Karnaugh, synthèse, schéma

- Le Half-Adder est circuit simple : un XOR et un AND.
  - Le XOR calcule la somme S (en anglais : Sum) (en effet un XOR réalise la fonction "addition modulo-2")
  - Le AND calcule le report C (en anglais : Carry) (en effet si les 2 entrées sont à 1, il y a report)

- Beaucoup d'auteurs utilisent le symbole suivant pour le Half-Adder :

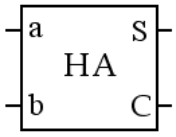


Figure 69 : Half-Adder. Symbole
Entrées : a, b ; Sorties : Somme S, Report (Carry) C

15.4.4 Cellule "Additionneur binaire avec reports (entrée et sortie)" = "1-bit Full-Adder"

- Entrées, Sorties, Fonctions

  La cellule complète "additionneur binaire" a 3 bits d'entrée :
  a, b, et C_{in} (= Carry in = entrée report)
  Cette cellule a une sortie Somme S, et une sortie C_{out} (Carry out, sortie report)

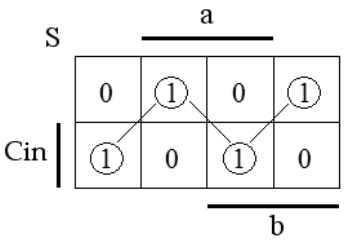
  Au § 15.4.4 , on connectera l'entrée report C_{in_i} à $C_{out_{i-1}}$ (la sortie Report de la cellule du rang précédent i –1).

  La table de vérité de la cellule i est la suivante :

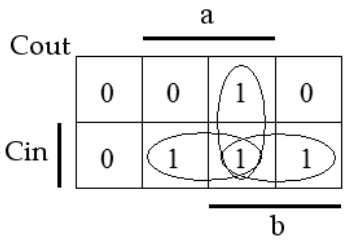
	Entrées			Sorties	
n	C_{in}	b	a	S	C_{out}
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Tableau 57 : Table de vérité des sorties (Somme et Report) d'une cellule Full-adder

- De la table de vérité on déduit les tables de Karnaugh de S et de C_{out} :



Structure en damiers
$\Rightarrow$ $S = (a \oplus b) \oplus C_{in} = a \oplus b \oplus C_{in}$



$C_{out} = a \,.\, b + b \,.\, C_{in} + C_{in} \,.\, a$

Figure 70 : Tables de Karnaugh des 2 sorties du Full-adder

- S est bien le résultat d'un XOR à 3 entrées
  (en effet un XOR réalise une addition modulo-2)

- C_{out} est simplement la fonction Majorité(a, b, C_{in})
  En effet il y a report dès que 2 au moins des 3 entrées valent 1.

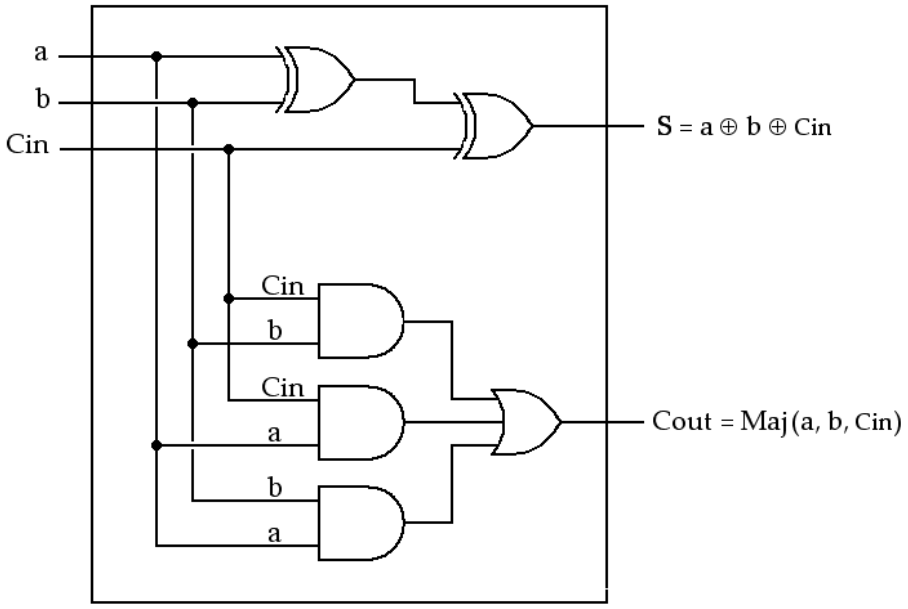


Figure 71 : 1-bit Full-Adder au moyen d'un XOR à 3 entrées et de la fonction Majorité

- Ce schéma est correct. Cependant la plupart des auteurs renseignent un autre schéma, qu'on peut par exemple obtenir avec le raisonnement suivant :

  On utilise 2 Half-Adders, l'un pour additionner a et b,
  et l'autre pour additionner son résultat avec Cin. Ceci donne le schéma :

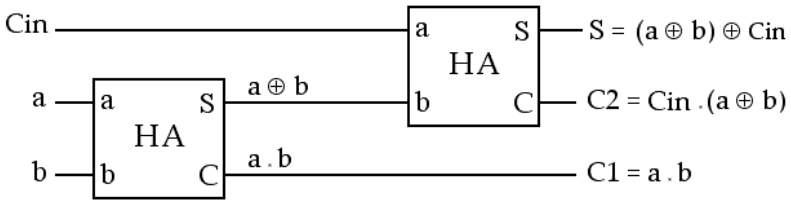


Figure 72 : Addition avec 2 Half-Adders. Ce schéma a 2 sorties report (Carry).

Ce schéma a 2 sorties report (Carry) : C1 et C2.

Mais si C1 = 1 : C1 = 1 $\Rightarrow$ a = b = 1 $\Rightarrow$ a $\oplus$ b = 0 $\Rightarrow$ C2 = 0
$\Rightarrow$ les 2 reports C1 et C2 ne peuvent pas être simultanément à 1
$\Rightarrow$ pour additioner les 2 reports, il suffit d'un OR :

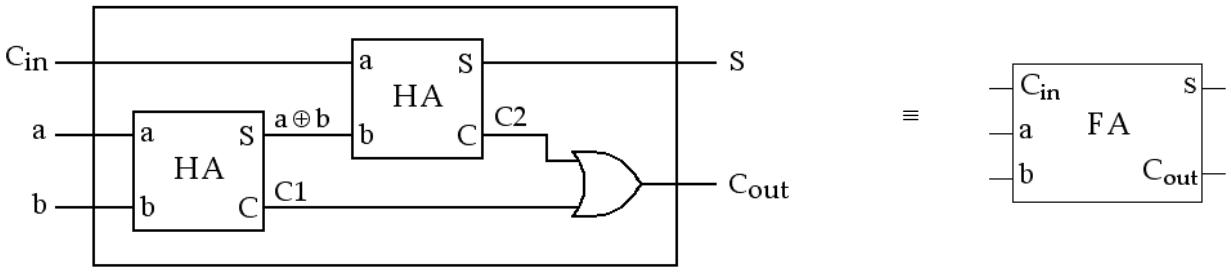


Figure 73 : Le "1-bit Full-Adder" est composé de 2 Half-Adders et d'un OR
Entrées : a , b , C_{in} . Sorties : S , C_{out} .
À droite : Logigramme usuel du 1-bit Full-Adder.

15.4.5 Additionneur arithmétique binaire itératif (= cascade) à n bits

- On fabrique un additionneur itératif à n bits
  au moyen de n cellules full-adder en cascade :

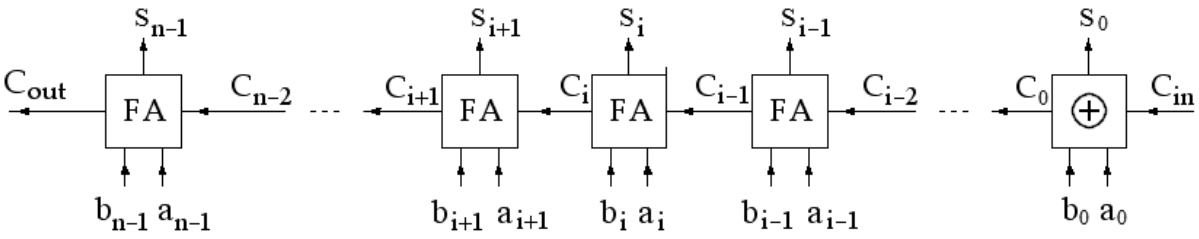


Figure 74 : Additionneur arithmétique itératif, à n bits

- On a placé les cellules de poids faibles à droite (comme lorsqu'on écrit un nombre),
  et le report se propage de droite à gauche : chaque cellule est dessinée avec
  - les variables d'entrée a et b en bas,
  - l'entrée report C_{in} côté droit, la sortie report C_{out} côté gauche
    (signal nommé C_i)
  - la sortie S (bit de somme) en haut.

  Le résultat de l'addition des mots a et b est
  - le mot formé par les n bits S_i ,
  - et le bit C_{out} , bit de sortie report de la cellule de gauche : $C_{out} = C_{n-1}$
  - Si le circuit additionneur est utilisé seul: connecter l'entrée report C_{in} à "0" .

- Exemple d'additionneur 4 bits du commerce : 74HC283

- Mise en cascade d'additionneurs :
  - Pour étendre le nombre de bits de l'additionneur,
    relier la sortie C_{out} d'un tel circuit à l'entrée C_{in} d'un circuit identique;
    (c'est pour permettre cette mise en cascade,
    que la cellule de rang 0 est aussi une cellule de Full-Adder et non de Half-Adder).
  - Connecter l'entrée report C_{in} du circuit additionneur de poids le plus faible à "0" .

15.5 Le comparateur (détecteur d'égalité)

- Nous désirons trouver une structure itérative pour un système qui indique si 2 nombres A et B de n bits sont égaux (E = 1) ou inégaux (E = 0).
- On décompose ce système en autant de cellules qu'il y a de bits:
  - on compare les 2 nombres bit à bit.

    Note : en général, on part du bit de poids le plus faible (bien que pour ce cas particulier, l'ordre n'a pas d'importance)
  - on transmet à la cellule suivante un signal qui vaut "1" s'il y a égalité, et qui vaut "0" s'il n'y a pas égalité.
- La sortie globale du système sera le dernier signal interstitiel (il s'agit d'un cas particulier : il n'y a pas de sortie parallèle, mais uniquement un signal interstitiel).

15.5.1 La cellule détecteur d'égalité sans entrée report

- Examinons d'abord le cas du bit de poids le plus faible. Pour ce bit, le problème ne comporte que 2 variables: les bits a_0 et b_0 de rang 0 des 2 nombres A et B.

  si $a_0 = b_0$, on a $E_0 = 1$ (égalité) ; si $a_0 \neq b_0$, on a $E_0 = 0$ (inégalité).
- La table de définition de la fonction égalité est donc la suivante:

Entrées b_0	Entrées a_0	Sortie E_0	Commentaire
0	0	1	égalité
0	1	0	
1	0	0	
1	1	1	égalité

Tableau 58 : Détecteur d'égalité de 2 bits : Table de vérité

- On en déduit la Table de Karnaugh et ensuite la synthèse :

E_0	a_0	
	1	0
b_0	0	1

Figure 75 : Détecteur d'égalité de 2 bits : Table de Karnaugh et circuit

Synthèse : La structure des 1 en damier indique soit un XOR des variables, soit son complément : ici il s'agit du complément du XOR (fonction XNOR) : $E_0 = \overline{a_0 \oplus b_0}$

15.5.2 La cellule détecteur d'égalité avec entrée report

- Pour les autres cellules, de rang quelconque i , nous devons tenir compte du résultat venant de la cellule de poids directement inférieur :

  l'entrée e_i de la cellule de rang i est la sortie E_{i-1} de la cellule précédente.

  - Si l'entrée $e_i = 0$, les 2 nombres sont différents, et il faut $E_i = 0$
  - Si $e_i = 1$, on retrouve la même table que celle donnée ci-dessus pour le rang 0:

Entrées			Sortie
$e_i = E_{i-1}$	b_i	a_i	E_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tableau 59 : Cellule détecteur d'égalité avec entrée report et sortie report Table de vérité

⇒

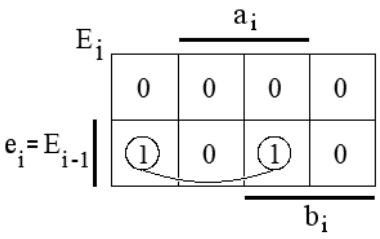


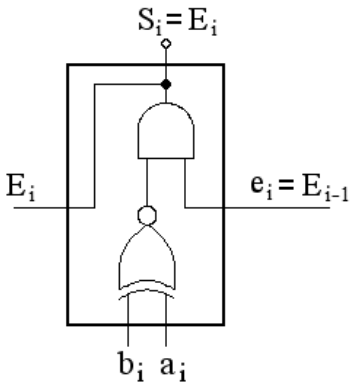
Figure 76 : Table de Karnaugh

- On a une structure en damier, mais uniquement sur la moitié inférieure de la table, correspondant au terme E_{i-1} .

  On a donc un AND entre ce terme et le OU EXCLUSIF trouvé plus haut, et indiqué par les 2 cercles reliés, si bien que:

$$E_i = e_i . (\overline{a_i \oplus b_i})$$

- De cette équation on déduit le schéma d'une cellule :



Le signal interstitiel E_i sert également de sortie parallèle.

Signification du signal E_i :

- si $E_i = 0$, les 2 nombres jusqu'au rang i sont différents
- si $E_i = 1$, les 2 nombres sont égaux jusqu'au rang i compris.

Figure 77 : Schéma d'une cellule cascadable de détecteur d'égalité (Bits de poids faibles : du côté droit)

15.5.3 Le comparateur (détecteur d'égalité) à n bits (structure en cascade)

- On réalise un détecteur d'égalité de 2 nombres binaires de n bits
  en cascadant n cellules.

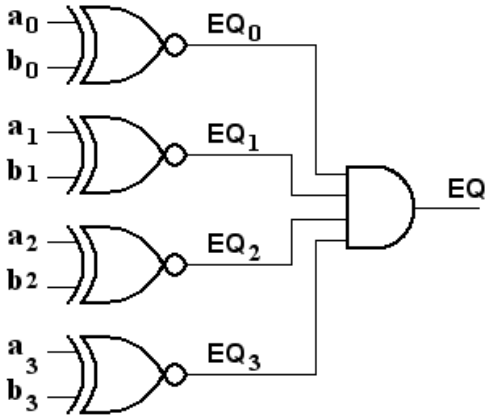
  Etant donné la signification du signal E_i :
  - si $E_i = 0$, les 2 nombres jusqu'au rang i sont différents
  - si $E_i = 1$, les 2 nombres sont égaux jusqu'au rang i compris,

  la sortie globale du système est la dernière sortie E_{n-1},
  qui indique si les 2 nombres de n bits sont égaux.

- Pour augmenter le nombre de bits, il suffit d'augmenter le nombre de cellules.
  C'est pourquoi il est intéressant d'utiliser le schéma complet également pour le rang 0,
  malgré qu'un simple XNOR convienne.

15.5.4 Le comparateur (détecteur) d'égalité à n bits (structure en parallèle)

- En structure parallèle, on relie les n XNOR aux entrées d'un (seul) AND à n entrées.
  Nous avons déjà vu ce schéma :



Réf. Figure 46 : Comparateur (détecteur d'égalité) à n bits (ici n = 4)

- Dans cet exemple très simple, les structures parallèle et itérative sont très proches.
  On voit cependant déjà que
  - le circuit parallèle est plus rapide (2 étages)
    mais peut être plus compliqué à étendre (ici il faut un AND à beaucoup d'entrées),
  - le circuit itératif est plus lent (le AND se fait par une cascade de AND à 2 entrées),
    mais plus facile à étendre: il suffit d'un fil.

15.6 Comparateur d'amplitude de 2 nombres binaires

- Nous désirons un système itératif qui indique,
  pour 2 nombres A et B de n bits, si A > B ou si A = B

15.6.1 Cellule comparateur sans entrée report

- Examinons d'abord le cas de deux nombres A et B chacun de 1 seul bit

Entrées		Sorties	
A	B	GT	EQ
0	0	0	1
0	1	0	0
1	0	1	0
1	1	0	1

Tableau 60 : Détecteur d'égalité de 2 bits : Table de vérité
GT ≡ "A Greater Than B" , EQ ≡ "A EQual B"

On en déduit : GT = $A \cdot \overline{B}$,

Pour EQ, on a comme auparavant : EQ = $\overline{A \oplus B} = \overline{A} \cdot \overline{B} + A \cdot B$

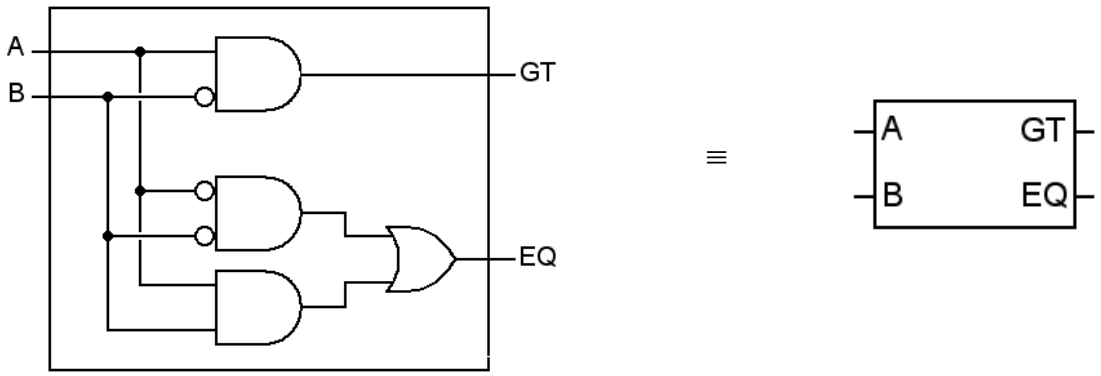


Figure 78 : Comparateur d'amplitude de 2 bits

15.6.2 Cellule comparateur avec entrées de report

- On compare tout d'abord les 2 bits de poids le plus fort
  - Si $a_n \neq b_n$, il n'y a pas besoin de tenir compte des résultats des comparateurs pour les bits de poids plus faible : le resultat pour GT est GT_n
  - Si $a_n = b_n$, il faut alors comparer les 2 bits de poids suivant : a_{n-1} et b_{n-1} ; si ces 2 bits sont à leur tour égaux, il faut alors comparer a_{n-2} et b_{n-2} ; et ainsi de suite…

  Dès lors, en fonction du bit EQ sortie du comparateur de a_n et b_n, on propage le résultat GT vers la droite à l'aide d'un Mux 2 vers 1 :

- On obtient la structure suivante

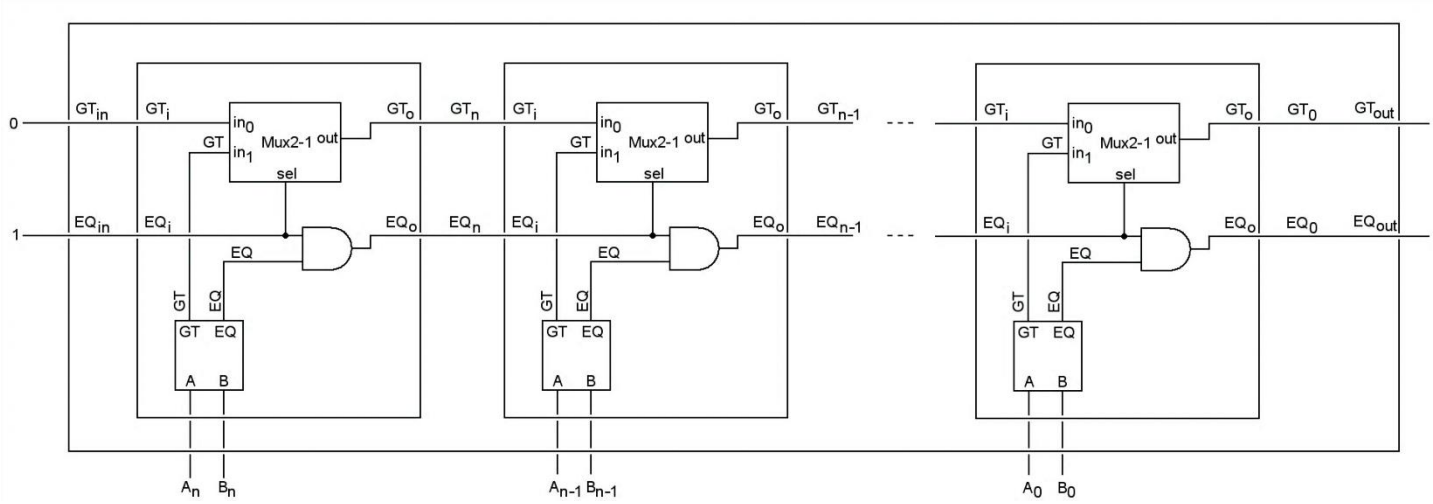


Figure 79 : Comparateur d'amplitude de 2 mots de n bits

Le bit de poids le plus fort est à gauche, les reports se font gauche vers droite.
Cellule la plus à gauche (de poids le plus fort) : pour permettre la mise en cascade, on a gardé la cellule complète $\Rightarrow$ on a connecté GT_{in} à 0, et EQ_{in} à 1

15.6.3 Cellule comparateur du commerce

- Voir par exmple 74HC85 : "4-bit Magnitude Comparator"
- Par rapport à l'exposé ci-dessus, ce C.I. dispose en plus d'une entrée LT_{in} et d'une sortie LT_{out} (LT ≡ "A Less Than B")

15.8 Le soustracteur, et l'additionneur/soustracteur arithmétiques binaires

- Pour créer un soustracteur *arithmétique*,
  on pourrait copier le raisonnement de l'additionneur arithmétique :
  - Le but du circuit soustracteur arithmétique est de calculer $S = A - B$;
    on considère seulement le cas des nombres positifs : $A \geq 0$, $B \geq 0$, $A \geq B$.
  - Se rappeler l'algorithme de soustraction arithmétique
    et la notion d'emprunt (au lieu de report) d'une unité du rang supérieur,
    lorsque la soustraction au rang actuel le nécessite.
  - Concevoir un circuit "Soustracteur sans retenue à l'entrée" (demi-soustracteur, Half-substractor), avec une sortie résultat (1 bit), et une sortie "emprunt" (1 bit).
    Concevoir ensuite une cellule "Soustracteur avec entrée retenue et sortie emprunt" ("Full-substractor"), réalisée au moyen de 2 half-substractors et d'un OR
  - Pour des nombres de n bits : mettre n cellules full-substractor en cascade

- Les schémas de l'additionneur et du soustracteur binaires arithmétiques étant proches,
  on pourrait imaginer un "additionneur-soustracteur", qui, suivant l'état d'une variable d'entrée binaire (commande) fera l'addition ou la soustraction de 2 entiers (positifs)

- Muni de ce circuit, on pourrait alors traiter des nombres *algébriques* ($< 0, = 0, > 0$)

- Dans d'anciens ordinateurs, les nombres algébriques étaient en effet représentés
  sous la forme de leur valeur absolue, et d'un bit de signe.
  Pour additionner ou soustraire 2 nombres codés ainsi, il fallait choisir l'opération à appliquer, vu que le circuit de calcul ne travaillait que sur des valeurs absolues.
  Il fallait aussi déterminer le bit de signe du résultat.
  Le circuit obtenu était assez lent.

- Aujourd'hui on ne représente plus les nombres en [valeur absolue et bit de signe].

  Au chapitre suivant, nous verrons la solution actuelle, plus efficace :
  En choisissant bien la façon de coder des nombres négatifs, on verra que
  - Le circuit pour calculer $m = -n$ est très simple
  - On n'a donc plus besoin d'un circuit soustracteur :
    pour calculer $s = n_1 - n_2$, on calcule $s = n_1 + m$ où $m = -n_2$
  - L'additionneur arithmétique
    (que nous avons conçu au départ pour traiter seulement des entiers ≥ 0)
    additionne en fait également correctement des entiers algébriques ($< 0, = 0, > 0$) :
    on n'aura en fait *pas besoin* d'un autre circuit d'addition ou de soustraction !!!

16 Représentation des entiers positifs ET négatifs dans une machine

16.1 Représentation des entiers ≥ 0 et < 0 dans une machine décimale à 3 chiffres

- Il y a 30 ans existaient des calculatrices mécaniques avec des roues dentées à 10 dents.
  Ces machines travaillaient sur des entiers naturels exprimés en décimal,
  par exemple avec 3 chiffres $\Rightarrow$ les nombres positifs étaient compris entre 0 et 999 .

- A l'aide d'un report mécanique, ces machines permettaient d'additionner 2 nombres :
  voici des exemples de calculs effectués par une telle machine :
  - 85 + 27 = 112 (correct, aucune remarque à faire)
  - 116 + 902 = 018 (faux, car le résultat correct est 1018)
    La machine ne retient que 3 chiffres, elle perd le report du millier :
    il y a "dépassement de capacité" (overflow).
    Nous allons voir qu'on peut exploiter les cas d'overflow.

- En fait, sans aucun changement, cette machine permet aussi de représenter des entiers négatifs, et de les additionner :

  Comme $\mathbf{10^3 = 1000}$ et comme 1000/2 = 500, **on convient ceci** :
  - On travaille seulement avec des entiers tels que $\mathbf{-500 \leq N \leq 499}$
  - Les entiers positifs N tels que $0 \leq N \leq 499$ sont représentés tels quels : K = N
    Exemple : le nombre N = 17 est représenté dans la machine par K = 17
  - Les nombres négatifs N tels que $-500 \leq N \leq -1$ sont représentés
    dans la machine par K = 1000+N
    (= 1000 – | N | = le "complément à 1000" de | N |)
    Exemple : le nombre –116 est remplacé par 1000 – 116 = 884

  $\Rightarrow$ l'addition de 2 nombres, positifs ou négatifs, fonctionne correctement !
  Exemple : 224 + (–116) : sur la machine on fait 224 + 884 = 1108, qui devient 108 (correct)

- Le complément à 999 d'un nombre entier | N| vaut : 999 – | N |
  Ce calcul ne nécessite aucun emprunt $\Rightarrow$ on complémente à 9 chaque chiffre séparé
  Exemple : 999 – 116 = 883

- Pour calculer le complément à 1000, on recommande la méthode suivante :
  - calculer le complément à 999 (une opération très simple à effectuer),
  - puis ajouter 1

  en effet : 1000 – | N | = 999 + 1 – | N | = (999 – | N |) + 1
  Exemple : 1000 – 116 = (999 – 116) + 1 = 883 + 1 = 884

- Conclusion : sur cette machine, pour calculer **224 – 116** :
  - On encode 224
  - On encode 116, on demande à la machine de le complémenter : le résultat est :
    - 883 (complément de 116 à 999), et
    - la machine positionne le report à 1 (en prévision de l'addition à venir)
  - On additionne. Le résultat est : 224 + 883 + 1 = 1108 avec overflow $\Rightarrow$ **108** (correct).

16.2 Représentation des entiers positifs et négatifs dans une machine binaire

- Commençons l'exposé avec des nombres de 8 bits
- Comme $2^8 = 256$ et comme $256/2 = 128$, on convient ceci :
  - *On représente seulement les nombres entiers N tels que $-128 \leq N \leq 127$*
  - *Soit K la "représentation", le code représentant l'entier N :*

- *Si $0 \leq N \leq 127 \quad \Rightarrow \quad K = N$ (on ne doit rien faire)*
  Note : le bit de poids fort de K vaut 0

Exemple : $N = 3_{déc}$ est représenté par : $K = 3_{déc}$

En binaire : $N = 00000011_{bin} \Rightarrow K = 00000011_{bin}$

- *Si $-128 \leq N \leq -1 \quad \Rightarrow \quad K = 256 - |N|$*
  En binaire : inverser chaque bit de $|N|$ et ajouter le nombre 1 [(1)]
  Note : le bit de poids fort de K vaut 1

Exemple : $N = -3_{déc}$ est représenté par : $K = 256 - 3 = 253_{déc}$

En binaire : $|N| = 00000011_{bin} \Rightarrow K = 11111100_{bin} + 1 = 11111101_{bin}$

- *Vocabulaire :*
  - *"Complémenter à 1" = inverser tous les bits d'un entier binaire*
  - *"Complémenter à 2" = Complémenter à 1, puis ajouter le nombre 1*

- *Circuits de complémentation :*

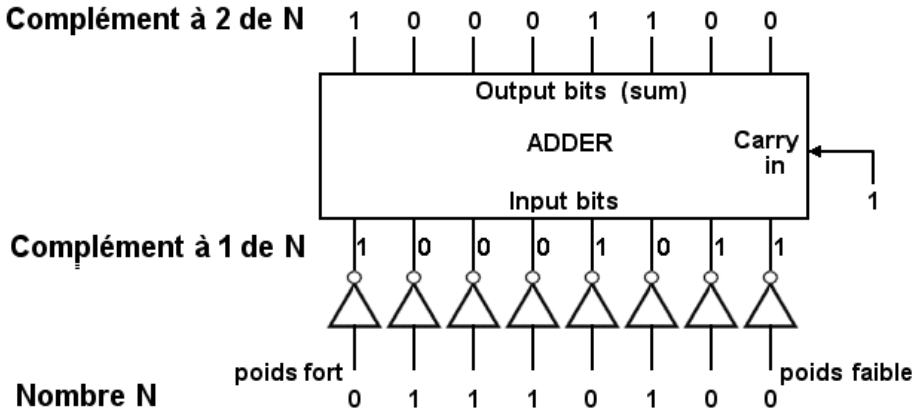


Figure 80 : Circuit de calcul du complément à 2 d'un nombre N de 8 bits (on inverse chaque bit, et au résultat on ajoute le nombre 1)

[(1)] En effet $K = 256 - |N| \quad \Rightarrow \quad K = 256 + N \quad \Rightarrow \quad K = (255 - |N|) + 1$

N	pour N ≥ 0 :	pour N ≤ 0 : | N |			pour N ≤ 0 : 255 – | N |		K		
déc	hex	déc	hex	bin	hex	Bin	déc	hex	bin
127	7F						127	7F	0111 1111
126	7E						126	7E	0111 1110
125	7D						125	7D	0111 1101
...	...						...	...	...
3	03						3	03	0000 0011
2	02						2	02	0000 0010
1	01						1	01	0000 0001
0	00	0	00	0000 0000	FF	1111 1111	0	00	0000 0000
–1		1	01	0000 0001	FE	1111 1110	255	FF	1111 1111
–2		2	02	0000 0010	FD	1111 1101	254	FE	1111 1110
–3		3	03	0000 0011	FC	1111 1100	253	FD	1111 1101
...		...	...	...			...	...	...
–126		126	7E	0111 1110	81	1000 0001	130	82	1000 0010
–127		127	7F	0111 1111	80	1000 0000	129	81	1000 0001
–128		128	80	1000 0000	7F	0111 1111	128	80	1000 0000

Tableau 61 : Nombres N tels que $-128 \leq N \leq 127$, et leur représentation K en 8 bits

- La méthode est utilisable avec un nombre quelconque de bits :
  les valeurs du nombre entier algébrique N représentables par un entier K de n bits sont :

$$\mathbf{-2^{n-1}\ \text{à}\ (2^{n-1}-1)}$$

Nombre de bits n	2^n	2^{n-1}	valeurs de N représentables
8	256	128	**– 128 à + 127**
16	65 536	32 768	**– 32 768 à + 32 767**
32	4 294 967 296	2 147 483 648	– 2 147 483 648 à + 2 147 483 647

Tableau 62 : Entiers signés représentables en 8, 16, 32 bits

16.2.1 Calcul du nombre N à partir de son code (sa représentation) K

- *Si le bit de poids fort de K vaut 0 $\Rightarrow$ N = K (on ne doit rien faire)*
- *Si le bit de poids fort de K vaut 1 $\Rightarrow$ N < 0 et | N | = 256 – K*
  (*et en binaire, il suffit à nouveau de complémenter K à 2)*

- Exemple :

  $K = 253_{déc}$ $\Rightarrow$ $|N| = 256 - 253 = 3_{déc}$ (c.-à-d. K = 253 représente $N = -3_{déc}$)
  En binaire : $K = 1111\,1101_{bin}$ $\Rightarrow$ $|N| = 0000\,0010_{bin} + 1 = 0000\,0011_{bin}$

16.3 Additionneur-soustracteur algébrique en complément à 2 ("complément vrai")

- Rappel : vocabulaire :
  - calcul arithmétique = avec des nombres > 0 ou $= 0$
  - calcul algébrique = avec des nombres > 0 , $= 0$, ou < 0

- En représentant les nombres < 0 en complément à 2, on réalise un additionneur-soustracteur *algébrique* avec un simple additionneur *arithmétique* :

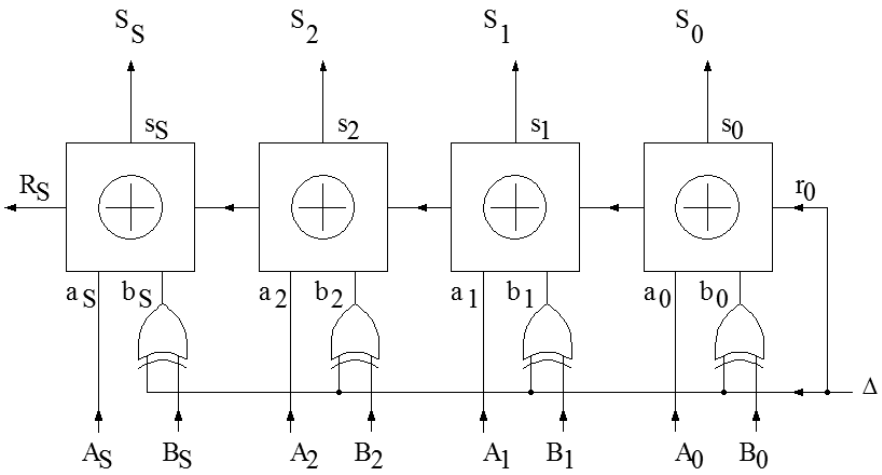


Figure 81 : Additionneur-soustracteur
Chaque bloc noté ⊕ est une cellule de full-adder (un simple additionneur arithmétique).
Les minuscules désignent les entrées et les sorties de l'additionneur arithmétique
Les majuscules désignent les entrées et les sorties de l'additionneur-soustracteur algébrique.

- Pou réaliser une addition, en place l'entrée de commande Δ à la valeur 0 :
  - le circuit calcule $S = A + B + r_0$, où l'entrée de report $r_0 = 0$

- Pour réaliser une soustraction, on place l'entrée de commande Δ à la valeur 1.
  Le circuit utilise la propriété (exprimée ici en 8 bits) : $A - B = A + (255 - B) + 1$:
  - on inverse tous les bits de B
    (inversion conditionnelle de chaque bit, au moyen d'un XOR).
  - on additionne une unité de poids le plus faible : l'entrée report $r_0 = 1$
    $\Rightarrow S = A + (255 - B) + 1$

16.4 EXERCICES

56. Complétez le tableau suivant (les nombres sont des entiers positifs)

Base = 10	Base = 16	Base = 2
		1101
11		
	F3	
		1101 1101
118		
605		
	AEF	

Tableau 63 : Exercice : Changement de base pour des entiers positifs

57. Trouvez le complément à 2 en 8 bits du nombre $1100\,1110_2$

58. $N = 11101100_{bin}$ est un entier signé écrit en complément à 2. Ecrivez N en décimal.

59. Trouvez le complément à 2 en 12 bits du nombre $10\,0100\,1011_2$

60. Est-il possible d'écrire les nombres entiers signés ci-dessous
en binaire en 8 bits en notation "en complément à 2" ? (Pourquoi ?). Si oui, faites-le.
    - $-35_{déc}$
    - $-168_{déc}$
    - $+215_{déc}$
    - $-105_{déc}$
    - $+105_{déc}$
    - $-21_{déc}$
    - $+207_{déc}$
    - $-63_{déc}$

61. Dans les expressions ci-dessous, les nombres sont signés
(notation des nombres négatifs en complément à 2, en 8 bits).
Réalisez les additions en binaire, puis vérifiez votre réponse en base 10 :
    - $0001\,0010_2 + 1000\,1011_2 =$
    - $1011\,0010_2 + 0010\,1011_2 =$
    - $0011\,0010_2 + 0111\,0011_2 =$

62. Remplissez un tableau inspiré du **Tableau 61** , mais cette fois pour les entiers k comprise entre **– 32 768 et + 32 767 inclus**

63. Réalisez et testez un Additionneur-soustracteur algébrique sur base d'un 74xx83A.

17 Représentation des nombres positifs fractionnaires en base B

- Nombre "fractionnaire" signifie : nombre non entier.
- Nous allons traiter séparément les nombres $N < 1$ et les nombres $N \geq 1$.
- La base est un nombre entier supérieur à 1.

17.1 Représentation en base B des nombres fractionnaires $0 \leq N < 1$

- Pour les nombres $0 \leq N < 1$, on garde le principe utilisé pour les nombres entiers positifs :

  > Si on déplace un chiffre d'un rang vers la droite, son poids est divisé par la base.
  > (Attention: à la droite de la virgule, les puissances de la base B sont négatives)

  Exemple :

  $0{,}847_{déc} = 8 \cdot 10^{-1} + 4 \cdot 10^{-2} + 7 \cdot 10^{-3} = 8 \frac{1}{10^1} + 4 \cdot \frac{1}{10^2} + 7 \cdot \frac{1}{10^3}$

- On peut utiliser une autre base que la base décimale :

  Soit un nombre N représenté dans une base B par la suite de chiffres:

  $N = 0, C_1 \ldots C_n{}_B$ avec $0 \leq C_i \leq B - 1$

  $\Rightarrow N = C_1 \cdot B^{-1} + C_2 \cdot B^{-2} + \ldots + C_n \cdot B^{-n} = \sum_{i=1}^{n} C_i \cdot B^{-i}$

  Exemple : $0{,}5642_8 = 5 \cdot 8^{-1} + 6 \cdot 8^{-2} + 4 \cdot 8^{-3} + 2 \cdot 8^{-4} = \frac{5}{8} + \frac{6}{64} + \frac{4}{512} + \frac{2}{4096}$

  $= 0{,}7316614_{déc}$

- Remarque :
  Dans une base, la représentation d'un nombre peut avoir peu de chiffres, et dans une autre base, la représentation du même nombre peut avoir une très grand nombre de chiffres, ou un nombre infini de chiffres.

  En pratique, on arrête la représentation du nombre à la décimale apportant la précision désirée.

  Exemples

  - le nombre $0{,}1_3 = 1 \cdot 3^{-1} = \frac{1}{3} = 0{,}33333\ldots_{déc}$
  - $N = 0{,}0111\ 0010\ 1110\ 1_2$

    Pour convertir N en base 10, on prend les puissances de 2 correspondant aux 1 :

    $N = 2^{-2} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-9} + 2^{-10} + 2^{-11} + 2^{-13}$

    $= \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{128} + \frac{1}{512} + \frac{1}{1024} + \frac{1}{2048} + \frac{1}{8192} = 0.4488525390625_{déc}$

17.2 Conversion base 10 → base B pour des nombres fractionnaires 0 ≤ N < 1

- Voici l'algorithme pour convertir un nombre dont la partie entière est nulle :

On multiplie le nombre en décimal à convertir par la base d'arrivée, ainsi que la succession des parties fractionnaires ainsi obtenues, jusqu'au rang correspondant à la précision désirée ou lorsque la partie fractionnaire est nulle.

La représentation du nombre dans la base B est la suite des parties entières des produits successifs, dans l'ordre direct où on les a trouvés.

- Pour obtenir une certaine précision en base B , jusqu'à quel rang m faut-il aller ?

  Pour obtenir une précision équivalente en décimal à $\frac{1}{10^n}$, en base B il faut aller jusqu'au rang m tel que $\boxed{\frac{1}{B^m} < \frac{1}{10^n}} \Rightarrow (B^m > 10^n)^{(1)} \Rightarrow m \cdot \log_{10} B > n \Rightarrow m > \frac{n}{\log_{10} B}$

- **Exemple 1 :**
  Convertir $0{,}4098_{déc}$ en octal, avec une précision (exprimée en décimal) de 10^{-4} :
  $\Rightarrow$ le rang m vaut : $m > 4 / \log_{10} 8 \cong 4 / 0{,}90309 \cong 4{,}4 \Rightarrow m = 5$

1	$0{,}4098 \times 8 =$ **3**, 2784
2	$0{,}2784 \times 8 =$ **2**, 2272
3	$0{,}2272 \times 8 =$ **1**, 8176
4	$0{,}8176 \times 8 =$ **6**, 5408
5	$0{,}5408 \times 8 =$ **4**, 3264 $\Rightarrow 0{,}4098_{déc} = 0{,}32164_8$

- **Exemple 2** : Convertir $0{,}5052_{déc}$ en binaire, avec une précision en décimal de 10^{-3} :
  $\Rightarrow$ calcul du rang : $2^9 = 512 < 10^3$ et $2^{10} = 1024 > 10^3 \Rightarrow m = 10$

1	$0{,}5052 \times 2 =$ **1**,0104
2	$0{,}0104 \times 2 =$ **0**,0208
3	$0{,}0208 \times 2 =$ **0,**0416
4	$0{,}0416 \times 2 =$ **0**,0832
5	$0{,}0832 \times 2 =$ **0**,1664
6	$0{,}1664 \times 2 =$ **0**,3328
7	$0{,}3328 \times 2 =$ **0**,6656
8	$0{,}6656 \times 2 =$ **1**,3312
9	$0{,}3312 \times 2 =$ **0**,6624
10	$0{,}6624 \times 2 =$ **1**,3248 $\Rightarrow 0{,}5052_{déc} = 0{,}1000\,0001\,01_{bin}$

  Plus rapide : **convertir le nombre en hex, puis convertir chaque chiffre hex en binaire** :
  Exemple : convertir 0,5052 en hex (3 chiffres), puis convertir en binaire, et limiter à m = 10 :

$0{,}5052 \times 16 =$ **8**,0832
$0{,}0832 \times 16 =$ **1**,3312
$0{,}3312 \times 16 =$ **5**,2992 $\Rightarrow 0{,}5052_{déc} \cong 0{,}815_{hex} = 0{,}1000\,0001\,0101_{bin}$
$= 0{,}1000\,0001\,01_{bin}$
(on limite à 10 chiffres binaires)

(1) L'inégalité stricte « > » peut être remplacée par l'inégalité non stricte « ≥ » ; cas particulier où B est une puissance entière de 10.

17.3 Conversion base 10 → base B pour des nombres fractionnaires $N \geq 1$

- Pour convertir un nombre (positif) $N \geq 1$ possédant une partie entière et une partie fractionnaire, d'une base quelconque en base 10, ou de la base 10 en une base quelconque, il faut :
  - traiter séparément les 2 parties,
  - puis réunir les 2 résultats ainsi obtenus, séparés par la virgule.

- Exemple: Convertir $2531{,}1325_{déc}$ en octal

  a) Partie entière

  2531 : 8 = 316 reste **3**
  316 : 8 = 39 reste **4**
  39 : 8 = 4 reste 7
  4 : 8 = 0 reste **4** $\Rightarrow$ $2531_{déc} = 4743_{oct}$

  b) Partie fractionnaire
  il faut 5 chiffres octaux pour avoir une précision décimale de 10^{-4} :
  $0{,}1325 \times 8 =$ **1**, 0600
  $0{,}0600 \times 8 =$ **0**, 4800
  $0{,}4800 \times 8 =$ **3**, 8400
  $0{,}8400 \times 8 =$ **6**, 7200
  $0{,}7200 \times 8 =$ **5**, 7600 $\Rightarrow$ $0{,}1325_{déc} = 0{,}10365_{oct}$

  c) Total
  $\Rightarrow$ $2531{,}1325_{déc} = 4743{,}10365_{oct}$

17.4 Représentation des nombres en virgule fixe

- Représenter des nombres entiers (exemple de nombres : 10573 et 2572) revient à placer la virgule à droite du dernier chiffre du nombre.

  Si l'ordre de grandeur des informations reste toujours le même, on peut placer la virgule à un endroit quelconque du mot, par exemple avec 2 chiffres décimaux à droite :
  Exemples de nombres : 5725,25 et 6375,60

  Cette représentation consistant à placer la virgule à un endroit quelconque (mais immuable) des nombres s'appelle la représentation "en virgule fixe".

- En virgule fixe, quel que soit l'emplacement choisi pour placer la virgule, les circuits de calcul restent les mêmes, la virgule ne faisant que modifier l'échelle.
  Entre les 2 chiffres séparés par la virgule, il y a le même rapport (2 en binaire, 10 en décimal ou en BCD par tranches de 4 bits) qu'entre 2 chiffres consécutifs quelconques.

  $\Rightarrow$ on pourra considérer que la virgule est à l'endroit qu'on veut, mais une fois cet endroit fixé, toute la suite du calcul devra respecter la même convention.

- Représenter des nombres en virgule fixe présente l'avantage que l'on garde toute la précision des nombres : la représentation en virgule fixe est souvent utilisée dans les programmes comptables où le résultat doit être exact au centime près quel que soit le nombre d'additions successives.

- Représenter les nombres en virgule fixe nécessite de concevoir le programme pour éviter les débordements. Ceci nécessite de l'attention.
  Pour l'exemple, considérons un calculateur traitant des nombres positifs de 3 chiffres décimaux (la virgule fixe est donc tout à droite).

  - Si on additionne les nombres 435 et 742, le circuit, conçu pour 3 chiffres, donne comme résultat de l'addition 177 au lieu des 1177 : le programmeur doit vérifier le signal de détection de débordement (OVERFLOW)
    et au besoin arrêter le calcul et signaler le dépassement.

  - Si on multiplie les 2 nombres, le résultat 322770 comporte 6 chiffres,
    c.-à-d. le double de ce que l'unité de calcul peut traiter.

    En général le circuit de multiplication est effectivement prévu pour fournir un résultat (produit) de longueur double[(1)]. Mais on doit quand même traiter le problème car le résultat de la multiplication sera sans doute utilisé comme opérande d'une opération ultérieure. Il faut donc choisir :

    - Soit, ce qu'on fait en virgule fixe, on garde les poids faibles (dans l'exemple, on garde les 3 chiffres de poids faibles) $\Rightarrow$ le résultat n'a de sens que si les nombres de départ sont suffisamment petits, en sorte que le produit est inférieur à la capacité de la machine (dans l'exemple : 999).

    - Soit on ne garde que les 3 chiffres de poids forts. On a une erreur d'arrondi, et il faut se souvenir du poids du résultat.

[(1)] Par exemple dans le µP Intel 8086, l'unité arithmétique du microprocesseur fournit les 2 demi-nombres (dans 2 registres à 8 ou à 16 bits selon le type d'opération), afin de ne rien perdre au résultat.

17.5 Virgule flottante (représentation par une mantisse et un exposant)

- Un problème de la représentation des nombres "en virgule fixe" est qu'elle nécessite un nombre exagéré de bits si on doit traiter des nombres d'ordres de grandeur très différents comme dans les calculs scientifiques : représenter des nombres aussi grands que la masse du soleil ($\sim 2\times10^{+30}$ kg) et aussi petits que la masse de l'électron ($\sim 1\times10^{-30}$ kg) en virgule fixe nécessiterait environ 200 bits[(1)] d'où un gaspillage et un surcoût en mémoire.

- La représentation "en virgule flottante", permet de représenter des nombres ayant des ordres de grandeur très différents, tout en conservant une longueur constante à leur représentation.

  Le principe est le même qu'avec la notation scientifique : on sépare le nombre en 2 parties, l'une donnant les chiffres significatifs, l'autre donnant l'ordre de grandeur.
  Les chiffres significatifs s'appellent la mantisse, l'ordre de grandeur s'appelle l'exposant.
  On exprime tout nombre réel N sous la forme :

  $$N = M \cdot b^E$$

  où b est la base de représentation (10 en décimal ou BCD, 2 en binaire),
  M est la mantisse, et E l'exposant.

  - La mantisse doit être normalisée, ce qui veut dire qu'elle doit être comprise entre 2 valeurs M_0 et $M_1 = \frac{M_0}{b}$: $\frac{M_0}{b} \le M < M_0$

    En décimal, on prendra une mantisse comprise entre $M_1 = 0{,}1$ et $M_0 = 1$.
    Certains choisissent une mantisse supérieure à 1, et inférieure à 10,
    mais la convention la plus répandue est celle donnée précédemment, à savoir:

    $$\mathbf{0{,}1 \le M < 1}$$

    Il est toujours possible de ramener M dans l'intervalle $[M_1, M_0[$
    en faisant varier l'exposant E puisque $M \cdot b^E = \frac{M}{b^x} \cdot b^{E+x}$

    Exemple : en décimal, le nombre 272 est représenté
    par la mantisse 0,272 et l'exposant 3 : $272 = 0{,}272 \cdot 10^3$

    En binaire, on prend une mantisse comprise aussi entre $0{,}1_{bin}$ et 1
    (rappelons que $0{,}1_{bin}$ vaut 1/2 ou $0{,}5_{déc}$).
    Rappelons que le 1 après la virgule a évidemment un poids différent selon la base:
    c'est seulement en décimal qu'il vaut 1/10 . En binaire, il vaut $1/2_{déc}$ soit $0{,}5_{déc}$. Avec ces conventions, voyons comment représenter les nombres de 0 à 15 en binaire pur en virgule flottante. Prenons l'exemple de 12. En notation entière
    (cas particulier de la virgule fixe), on a évidemment $12_{déc} = 1100_{bin}$.
    En virgule flottante, on a : $12 = 0{,}1100 \times 2^{100}$ (car $100_{bin} = 4$ et $2^4 = 16$).

---

(1) $10^3 = 1000 \cong 2^{10} \Rightarrow 10^{+30} \cong 2^{100} \Rightarrow$ pour écrire 10^{+30} en binaire, il faut environ 100 chiffres (bits)
De même 10^{-30} s'écrit en binaire avec environ 100 bits après la virgule ;
$\Rightarrow$ il faut environ 200 bits pour représenter ces nombres en virgule fixe.

17.6 Dynamique et précision relative

- Cette représentation en virgule flottante permet de traiter des nombres ayant des ordres de grandeur très différents, limités par le nombre de chiffres de l'exposant E .
    - Ainsi, en binaire, si l'exposant E a 8 bits, on peut manipuler des nombres de $0{,}5 \times 2^{-128}$ à $1 \times 2^{+127}$ (si on adopte pour l'exposant le code complément à 2).
- La précision relative de la représentation est fixée par le nombre de bits de la mantisse.
    - Évidemment, le problème du dépassement subsiste: il reste possible qu'un résultat, même intermédiaire, dépasse les capacités de la représentation. Mais les limites peuvent être repoussées assez loin, avec un nombre raisonnable de bits :

      Si nous reprenons l'exemple des nombres dont la gamme va de 10^{-30} à 10^{+30},
      pour l'exposant il faut 6 bits (pour représenter en complément à 2 un exposant de –32 à +31) et pour la mantisse, par ex. 3 octets
      soit 24 bits, donnent une précision relative de 2^{-23} (1 bit sert pour le signe de la mantisse) soit environ $1/8000000$, ce qui suffit pour beaucoup de calculs.

      On a donc au total 30 bits, alors qu'en virgule fixe on a besoin d'environ 200 bits.

17.7 Implémentation : généralités

- Il existe de nombreuses représentations possibles en virgule flottante, selon les conventions choisies pour la mantisse et l'exposant : mantisse signée ou non, exposant signé ou non, système binaire ou système décimal, ou encore d'autres (il existe par exemple une représentation où l'exposant est codé sur 7 bits, mais est considéré comme exposant de 16).
  Dans tous les cas, la structure de la représentation d'un nombre signé en virgule flottante est la suivante:

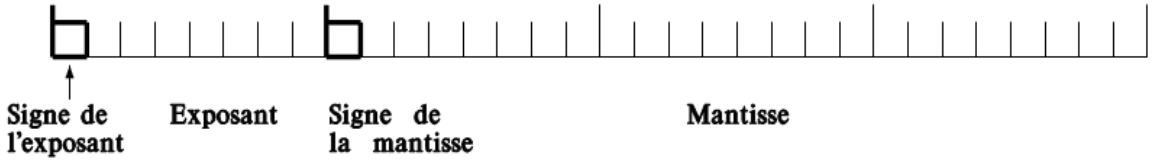


Figure 82 : Représentation d'un nombre signé en virgule flottante
Le signe du nombre est celui de la mantisse.

- Une unité de calcul en virgule flottante (FPU – Floating Point Unit) est compliquée, et la bibliothèque de sous-programmes manipulant des nombres en virgule flottante est assez compliquée. En effet, il faut souvent normaliser les grandeurs intervenant dans les opérations.
    - Pour effectuer une multiplication : on multiplie les mantisses, on additionne les exposants :

      $$A = M_A \cdot 2^{E_A} \qquad B = M_B \cdot 2^{E_B} \qquad A \times B = M_A \cdot M_B \cdot 2^{E_A+E_B}$$

      puis on normalise le résultat en appliquant $M \times b^E = \frac{M}{b^x} \times b^{E+x}$ jusqu'à ce que le 1er bit de la mantisse soit à 1 (mantisse comprise entre 0,1 et 1).
    - Pour effectuer une division, on divise les mantisses, on soustrait les exposants, puis on normalise le résultat.

      $$A = M_A \cdot 2^{E_A} \qquad B = M_B \cdot 2^{E_B} \qquad \frac{A}{B} = \frac{M_A}{M_B} \cdot 2^{E_A-E_B}$$

- Pour les additions et les soustractions, il faut d'abord "dénormaliser" l'exposant du plus petit des nombres, afin que les mantisses correspondent bit à bit aux mêmes poids. Puis on additionne les mantisses.
  - si le résultat ne déborde pas, la valeur du résultat est la mantisse obtenue avec l'exposant de départ du plus grand nombre,
  - si le résultat déborde, il faut renormaliser en décalant la mantisse et en augmentant l'exposant.

- Exemples:
  - Commençons par quelques transformations de nombres, de virgule fixe en virgule flottante normalisée

		Mantisse	Exposant
+ 11,01101	⇒	+ 1101101	+ 010
– 1110,100	⇒	– 1110100	+ 100
+ 0,000110	⇒	+ 1100000	- 011
– 0,010101	⇒	– 1010100	- 001

  - Exemples d'addition : $A = M_A \cdot 2^{E_A}$ $B = M_B \cdot 2^{E_B}$

    Supposons que $E_A > E_B$ et que $E_A = E_B + n$ (donc avec n positif).

    $A = M_A \cdot 2^{E_A}$ $B = M_B \cdot 2^{E_A - n} = M_B \cdot 2^{-n} \cdot 2^{E_A} = \frac{M_B}{2^n} \cdot 2^{E_A}$

    $\Rightarrow A + B = (M_A + \frac{M_B}{2^n}) \cdot 2^{E_A}$

    Donc, pour additionner 2 nombres en virgule flottant, il faut:
    1. comparer les exposants et calculer leur différence n.
    2. diviser la mantisse du plus petit par 2^n,
       ce qui correspond à n décalages vers la droite (poids faibles).
    3. additionner la mantisse du premier à la nouvelle mantisse du second.
    4. Si le résultat ne déborde pas, on garde la mantisse et l'exposant.
       Si le résultat déborde, on décale la mantisse d'un rang vers la droite,
       et on augmente l'exposant d'une unité.

- Nous en resterons là pour montrer le type de problèmes qui se présentent pour traiter des nombres en virgule flottante, le sujet étant très vaste (une bonne bibliothèque de calcul doit aussi être optimisée)

- Pour terminer, rappelons que l'ordinateur ne traite que des 0 et des 1.
  Alors, comment fait-il la différence entre 0100 0001 signifiant A et 0100 0001 valant 65 ?

  En fait, c'est le programme qui doit être conçu pour traiter soit des nombres, soit des caractères : par exemple le programme doit pouvoir traiter des nombres binaires, des nombres décimaux, entiers ou fractionnaires, signés ou non signés, en simple ou en double précision, en virgule fixe ou en virgule flottante, ...

  C'est au programmeur d'écrire le programme afin qu'il soit adapté au format des variables.
  D'ailleurs les langages (Basic, C, C++ par exemple) demandent
  en général de déclarer le type des différentes variables (voir cours d'informatique).

17.8 Implémentation standard : la norme ANSI/IEEE 754-1985

- Un nombre à virgule flottante, ou nombre réel, comprend 2 parties et 1 signe :
  - Le **signe** indique si le nombre est positif ou négatif
  - La **mantisse** est la partie qui donne les chiffres significatifs du nombre,
  - L'**exposant** est la partie qui désigne la quantité de rangs avec laquelle la virgule (décimale ou binaire) doit être décalée.

Signe S	Exposant E	Mantisse F

- Exemple : Le nombre décimal 287,7283 est codé en décimal sous la forme $0{,}2877283 \times 10^3$
  - Signe : +
  - Mantisse : 2877283
  - Exposant : 3

- Pour les **nombres à virgule flottante binaires**, la norme ANSI/IEEE 754-1985 définit 3 variantes :

Nom du format	Nombre de bits			
	Total	**Signe**	**Exposant**	**Mantisse**
Simple	32 bits	1 bit	8 bits	23 bits
Double	64 bits	1 bit	11 bits	52 bits
Étendu	80 bits	1 bit	15 bits	64 bits

Tableau 64 : Nombres de bits d'un réel en format IEEE

- Pour éviter des représentations différentes du même nombre, la mantisse est normalisée. Dans la convention la plus courante, un nombre binaire normalisé différent de zéro a la forme: $\pm 1.bbb...b \times 2^{\pm e}$

  Comme, sous cette forme, le bit le plus signifiant est toujours égal à 1, il n'est pas nécessaire de le coder: il est implicite

- En général, et afin de permettre la représentation des exposants négatifs, l'exposant est représenté de façon biaisée: une constante, le biais, doit être soustraite de la valeur dans le champ pour obtenir la vraie valeur de l'exposant: champ exposant = exposant + biais

  Typiquement, la valeur du biais est $2^{k-1}-1$, où k est le nombre de bits du champ de l'exposant. En format simple (32bits, dont 8 bits pour l'exposant), ce biais est de $2^7-1= 127$

  Toutefois, les deux valeurs extrêmes du champ exposant sont réservées pour des cas particuliers:
  00…00: pour les nombres non normalisés ($0 \leq X < 1$)
  11..11: pour infini (positif et négatif) et NaN (not a number)

- En format simple, la représentation d'un nombre binaire suit donc l'équation suivante :

$$\text{Nombre} = (-1)^S \cdot (1+F) \cdot (2^{E-127})$$

Voici un exemple en format simple (32 bits) : $(-5777{,}625)_{10}$

1. En binaire, ce chiffre donne : 1 0110 1001 0001,101 ou $1{,}0110\,1001\,0001\,101 \times 2^{12}$
2. Comme il s'agit d'un nombre négatif, le signe vaut : S = 1
3. Les 8 bits de l'exposant représentent un exposant « polarisé », obtenu en additionnant 127 à l'exposant réel. De ce fait, il n'est plus nécessaire d'utiliser un bit de signe pour les exposants négatifs. $E = 12 + 127 = 139_{déc} = 1000\ 1011_{bin}$
4. La mantisse, exprimée sur 23 bits, compte en fait 24 bits. Ceci est dû au fait que le bit de poids le plus significatif (rang 0) de tout nombre binaire est toujours 1. Par conséquent, il est convenu qu'un 1 occupe le rang 0, bien qu'on n'écrit pas ce 1 dans la chaîne des bits.
   M = 011 0100 1000 1101 0000 0000
5. Le nombre final, en format simple, sera stocké en mémoire sous la forme :

1	1000 1011	011 0100 1000 1101 0000 0000

- Nombres non normalisés : avec la représentation normalisée des nombres réels, le bit à gauche du point décimal est toujours égal à 1, implicitement. Il est donc impossible de représenter la valeur 0.0 de façon normalisée

  Pour résoudre ce problème, certains nombres sont non normalisés: dans ces cas, le bit à gauche du point décimal est égal à 0

  Les nombres non normalisés sont ceux dont le champ de l'exposant est égal à 0. Les nombres non normallisés différents de 0.0 sont utilisés pour représenter de très petites valeurs, proches de 0.0 Normalement, la valeur de l'exposant d'un nombre non normalisé devrait être -biais. Toutefois, pour assurer une meilleure transition entre les nombres normalisés et les non normalisés, l'exposant est calculé dans ces cas comme 1-biais

  Exemple de nombres non normalisés :

  +0.0 est représenté par :

0	00000000	0000 0000 0000 0000 0000 000

  -0.0 est représenté par :

1	00000000	0000 0000 0000 0000 0000 000

17.9 EXERCICES

64. Coder selon la norme ANSI/IEEE 754-1985 les nombres suivants :

    - $32\ ,450\ 10^{4}_{\text{déc}}$
    - $-3,150_{\text{déc}}$
    - $56,125\ 10^{-6}_{\text{déc}}$

65. Décoder les nombres binaires suivants :
    - 1 10000010 00110000000000000000000
    - 0 00000000 00000000000000000000000
    - 0 11111111 00000000100000000000100

66. Donner les plus petit et plus grand nombre normalisé positifs en format simple

18 Caractéristiques électriques des niveaux logiques

18.1 Introduction

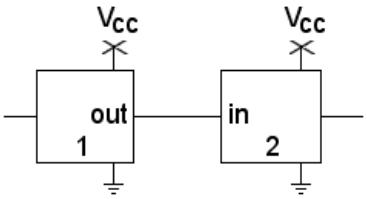


Figure 83 : Le C.I.1 pilote le C.I.2

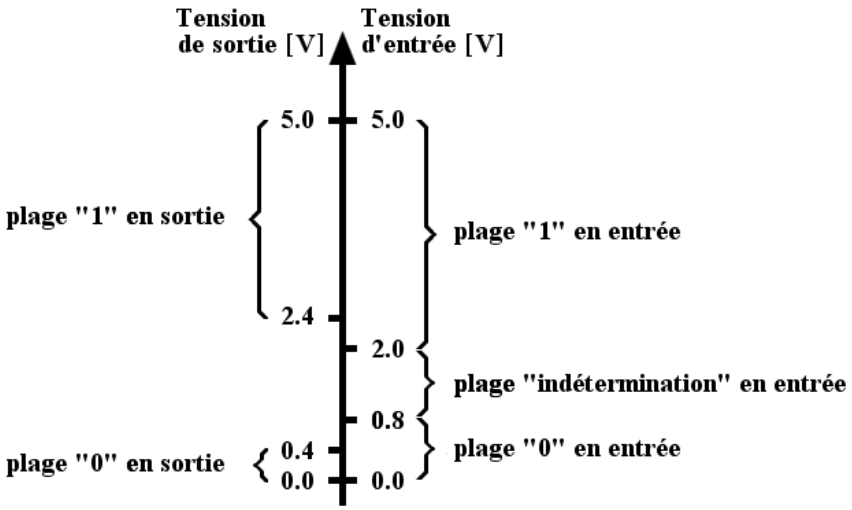


Figure 84 : Circuits TTL: plage de tensions en sortie et en entrée

- Soit un circuit n° 1 dont la sortie est connectée à (pilote) l'entrée du circuit n° 2.

  L'idée est de comparer la tension **exigée par le circuit 2**, à celle **garantie par le circuit 1 :**

18.2 Fonctionnement au niveau haut

- Si le circuit 1 affiche un 1 logique, on veut que le circuit 2 le détecte correctement.
  Pour cela, les fabricants *définissent en fait 2 niveaux hauts* : celui en sortie, et celui en entrée :

  - **V_{OH}** = tension de sortie au niveau haut (Output High)
    Exemple: à sa sortie, un circuit TTL type 7400 représente l'état 1 par $V_{OH} \geq$ 2.4V. Plus précisément, pour tous les chips le fabricant garantit

    $$V_{OH} \geq V_{OH_{min}} = 2.4V$$

    $V_{OH_{min}}$ = tension de sortie *minimum garantie* par le fabricant (*guaranteed* value)

  - **V_{IH}** = tension d'entrée au niveau haut (Input High)
    Exemple: à son entrée, un circuit TTL interprète $V_{IH} \geq 2V$ comme l'état 1

    Plus précisément, afin que le chip n°2 interprète la tension à son entrée comme l'état 1, le fabricant exige que la tension d'entrée respecte

    $$V_{IH} \geq V_{IH_{min}} = 2V$$

    $V_{IH_{min}}$ = tension d'entrée *minimum requise* par le fabricant (*required* value)

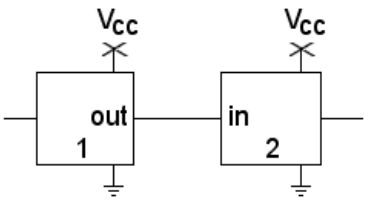
- $V_{OH_{min}} > V_{IH_{min}}$ (dans l'exemple 2.4V > 2.0V)

  $\Rightarrow$ Lorsque la sortie du circuit n°1 est au niveau haut,
  le circuit n° 2 interprète correctement la tension appliquée à son entrée comme un "1"

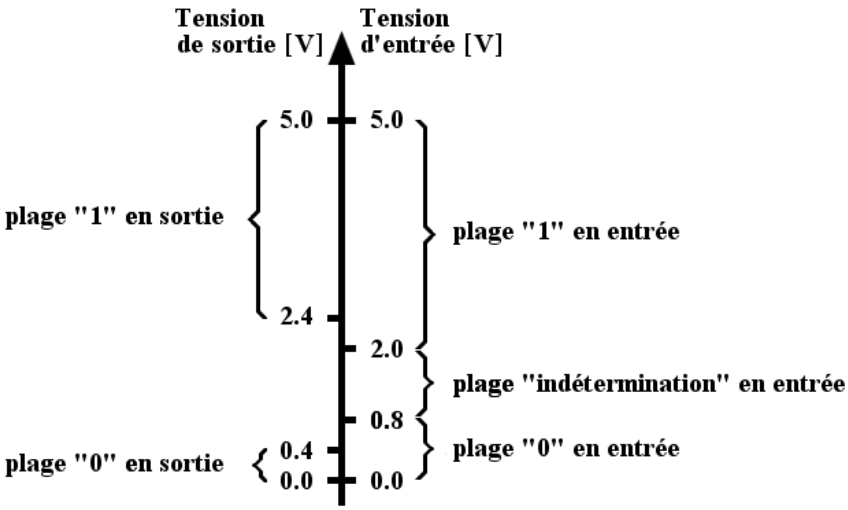
  $\Rightarrow$ Conclusion : ces 2 circuits sont compatibles au niveau haut.

18.3 Fonctionnement au niveau bas

- Reprenons les mêmes figures :



Réf. Figure 83 : Le C.I.1 pilote le C.I.2



Réf. Figure 84 : Circuits TTL: plage de tensions en sortie et en entrée

- Si le circuit 1 affiche un 0 logique, on veut que le circuit 2 le détecte correctement.
  Pour cela, les fabricants *définissent en fait 2 niveaux bas* : celui en sortie, et celui en entrée :

  - **V_{OL}** = niveau de sortie à l'état bas (Output Low)
    Exemple: à sa sortie, un circuit TTL représente l'état "0" par $V_{OL} \leq 0.4V$

    Plus précisément, pour tous les chips le fabricant garantit

    $$V_{OL} \leq V_{OL_{max}} = 0.4V$$

    $V_{OL_{max}}$ = tension de sortie *maximum garantie* par le fabricant (*guaranteed* value)

  - **V_{IL}** = niveau d'entrée à l'état bas (Input Low)
    Exemple: à son entrée, un circuit TTL interprète $V_{IL} \leq 0.8V$ comme l'état 0

    Plus précisément, afin que le chip n°2 interprète la tension à son entrée comme l'état 0 , le fabricant exige que la tension d'entrée respecte

    $$V_{IL} \leq V_{IL_{max}} = 0.8V$$

    $V_{IL_{max}}$ = tension *maximum requise* par le fabricant (*required* value)

- $V_{OL_{max}} < V_{IL_{max}}$ (dans l'exemple $0.4V < 0.8V$)

  $\Rightarrow$ Lorsque la sortie du circuit n°1 est au niveau bas,
  le circuit n° 2 interprète correctement la tension appliquée à son entrée comme un 0 :

  $\Rightarrow$ Conclusion : ces deux circuits sont compatibles au niveau bas.

18.4 Plage (zone) d'indétermination

- En entrée, les plages de tension "0" et "1" sont séparées par la "zone d'indétermination". Dans l'exemple (circuit TTL), la plage d'indétermination en entrée est : 0.8V à 2.0V

- Si la tension d'entrée d'un circuit est dans la zone d'indétermination, le fabricant
  - garantit que le circuit ne sera pas détruit
  - ne garantit pas le fonctionnement du circuit :
    - sa sortie peut au hasard choisir l'état 0, ou l'état 1, ou prendre toute valeur analogique entre 0 et Vcc (par ex. Vcc/2), ou osciller
    - le courant consommé par le circuit peut beaucoup augmenter (facteur 10).

18.5 Notation sans les indices

- Au lieu de $V_{OH_{min\ garanti}}$, $V_{IH_{min\ requis}}$, $V_{OL_{max\ garanti}}$, $V_{IL_{max\ requis}}$

  on écrit en bref V_{OH}, V_{IH}, V_{OL}, V_{IL}.

18.6 Conditions de charge – Fan-out d'un circuit TTL

- Le fabricant spécifie les valeurs V_{OH}, V_{IH}, V_{OL}, V_{IL} pour tous les circuits [(1)],
  mais seulement à condition que l'utilisateur respecte :
  - les conditions de température.
    Par ex. la température doit être comprise entre 0°C et 70°C (voir la datasheet).
  - les conditions de charge des circuits, c.-à-d. le courant maximum acceptable à la sortie d'un chip.
    Nous étudierons ces conditions de charge plus tard, mais
    si le circuit contient *uniquement des circuits TTL d'une même famille*,
    alors vous pouvez appliquer la règle suivante, très simple d'emploi :

La "sortance" (le "fan-out") d'un circuit TTL vaut 10

    c.-à-d. : vous pouvez connecter une sortie TTL à maximum 10 entrées TTL
    (de la même famille).

[(1)] Le fabricant spécifie les valeurs extrêmes de V_{OH}, V_{IH}, V_{OL}, V_{IL} pour les plus mauvais circuits

18.7 Notion de marge dans la vie courante

- Vous voulez acheter une carte GSM prépayée au magasin. Vous ne connaissez pas le prix, mais les cartes de ce fournisseur coûtent au maximum 20,00 €.
  $\Rightarrow$ si vous emportez 20,00 €, vous êtes certain de pouvoir l'acheter.
- Après réflexion, vous décidez d'emporter un peu plus : 21,00 €.
  La différence, 21,00 – 20,00 = 1,00 € est votre réserve, protection, "marge", "immunité" :
  grâce à cette marge, vous pourrez acheter la carte même en cas d'imprévu
  (par exemple le parking près du magasin est devenu payant) …

  … du moins pour un imprévu de taille limitée : si le parking coûte moins de 1 €, pas de problème !

18.8 Marge de bruit (Immunité au bruit) au niveau haut

- Reprenons le schéma d'interconnexion de 2 circuits et ajoutons une perturbation :

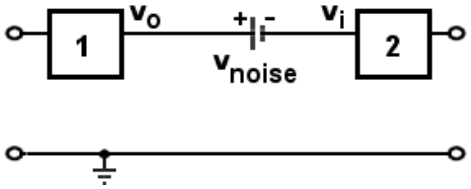


Figure 85 : Signal au niveau haut, perturbé vers le bas
La sortie du circuit 1 est connectée à l'entrée du circuit 2,
mais un parasite (assimilable à une source de tension)
diminue la tension d'entrée du circuit 2

- La tension de sortie du circuit 1 vaut au minimum $V_O = 2.4\,V$.
- Hélas V_I , tension d'entrée du circuit 2, est plus petite.
  En effet, $V_I = V_O - v_{noise}$

○ Définition :

La marge de bruit au niveau haut est

l'amplitude v_{noise} d'une tension parasite qu'on peut soustraire de V_O ,

sans que la tension d'entrée $V_I = V_O - v_{noise}$ descende

au point d'entrer dans la zone d'indétermination :

Immunités au bruit (= marge de bruit) au niveau haut : $\boxed{\textbf{Marge}_H = V_{OH} - V_{IH}}$

Dans l'exemple, l'immunité au bruit vaut $\text{Marge}_H = 2.4 - 2 = 0.4\,V$

(Représentation graphique : voir **Figure 87**)

En d'autres mots, la marge de bruit au niveau haut est
l'amplitude maximale admissible d'un parasite négatif [1] qui, lorsqu'il est superposé à V_{OH},
permet au signal d'entrée d'être encore interprété correctement (comme un niveau haut).

Note: en utilisant les notations complètes : $\textbf{Marge}_H = V_{OH\,\text{min garanti}} - V_{IH\,\text{min requis}}$

[1] Considéré comme la configuration défavorable.

18.9 Marge de bruit (Immunité au bruit) au niveau bas

- Reprenons le schéma d'interconnexion de 2 circuits et ajoutons cette fois une perturbation qui va faire monter V_I :

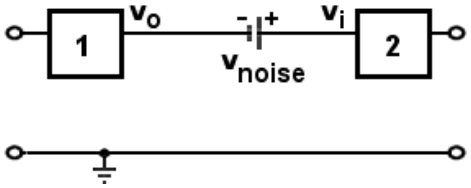


Figure 86 : Signal au niveau bas, perturbé vers le haut.
La sortie du circuit 1 est connectée à l'entrée du circuit 2,
mais un parasite (assimilable à une source de tension)
augmente la tension d'entrée du circuit 2

- La tension de sortie du circuit 1 vaut au maximum $V_O = 0.4\,V$.
- Hélas, la tension d'entrée du circuit 2, V_I est plus grande
  En effet $V_I = V_O + v_{noise}$
- Définition :

  La marge de bruit au niveau bas est

  l'amplitude v_{noise} d'une tension parasite qu'on peut ajouter à V_O ,
  sans que la tension d'entrée $V_I = V_O + v_{noise}$ augmente
  au point d'entrer dans la zone d'indétermination :

Immunité au bruit (= marge de bruit) au niveau bas :	$\mathbf{Marge_L = V_{IL} - V_{OL}}$
Dans l'exemple (circuits TTL) l'immunité au bruit vaut :	$Marge_L = 0.8 - 0.4 = 0.4\,V$

  (Représentation graphique : voir **Figure 87** page suivante)

  En d'autres mots, la marge de bruit au niveau bas est

  l'amplitude maximale admissible d'un parasite positif [(1)] qui, s'il est superposé à V_{OL},
  permet au signal d'être d'encore interprété correctement (comme un niveau bas) à l'entrée

- Note : en utilisant les notations complètes : $\mathbf{Marge_L = V_{IL_{max\ requis}} - V_{OL_{max\ garanti}}}$

[(1)] Afin de considérer le cas défavorable.

18.9.1 Immunité au niveau haut, immunité au niveau bas

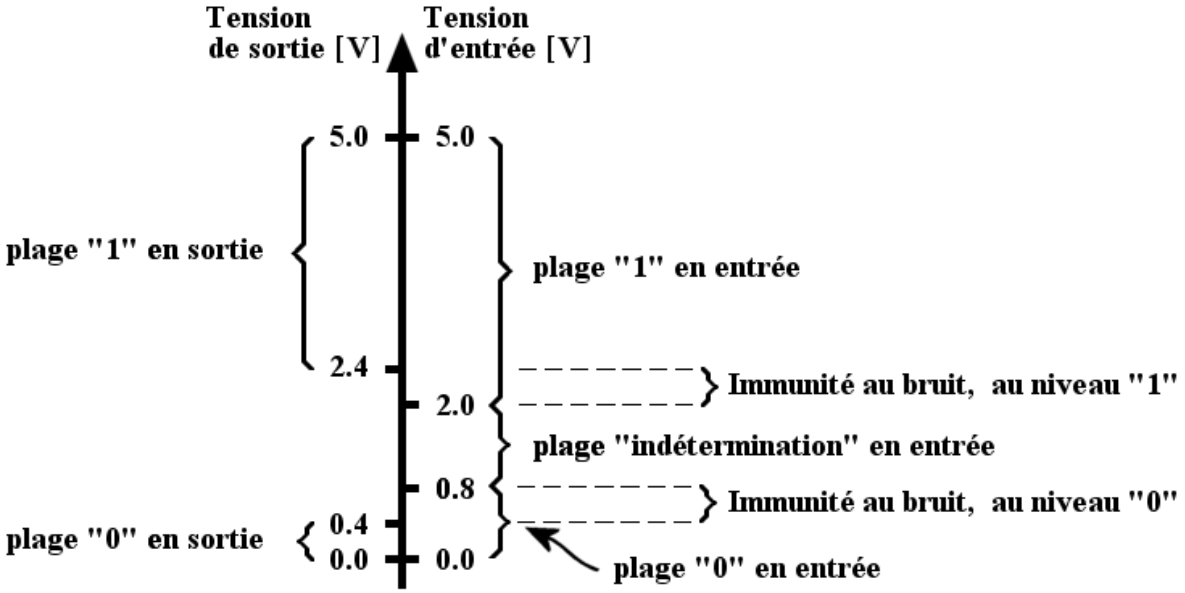


Figure 87 : Circuit logique TTL:
immunité au bruit au niveau haut, au niveau bas

- D'une famille de circuits logique à une autre (DTL, TTL, STTL, LSTTL, CMOS, HCMOS, LVCMOS, ECL, ...), les valeurs de V_{OH}, V_{IH}, V_{OL}, V_{IL} sont différentes, et donc :
  - lorsqu'on utilise un circuit, il faut consulter la datasheet du circuit
  - les marges de bruit sont différentes.
- En TTL, les deux marges de bruit (celle au niveau bas et celle au niveau haut) sont égales ; pour d'autres familles logiques, par exemple HCMOS et les variantes LVCMOS (Low voltage CMOS) , les 2 marges de bruit diffèrent l'une de l'autre.

18.10 Signaux dynamiques: durée du passage dans la zone d'indétermination

- Jusqu'à présent nous avons considéré des signaux statiques : constants au cours du temps, en permanence ou bien à l'état logique 0, ou bien à l'état logique 1.
  Mais que se passe-t-il lorsqu'un signal change de valeur ?

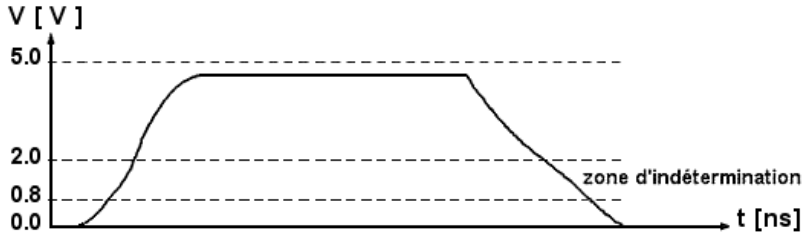


Figure 88 : Passages dans la zone d'indétermination
Un signal qui passe de 0 à 1 ou de 1 à 0
passe forcément dans la zone d'indétermination en entrée !

18.10.1 Durée max. acceptable d'un flanc de commutation en entrée

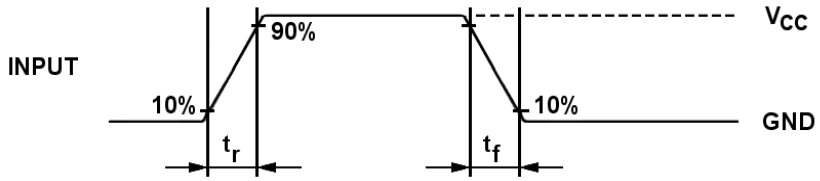


Figure 89 : Flancs de commutation de la tension V_I à l'entrée d'un circuit

- L'utilisateur d'un circuit logique doit faire en sorte que la tension d'entrée V_i d'un circuit passe suffisamment vite dans la zone d'indétermination : Le fabricant spécifie
  - la durée max. tolérée pour un flanc montant en entrée : t_r (= rise time)
  - la durée max. tolérée pour un flanc descendant : t_f (= fall time)

18.10.2 Durée maximum garantie d'un flanc en sortie (guaranteed output transition time)

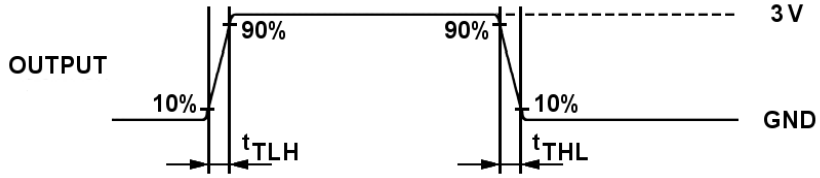


Figure 90 : Flancs de commutation garantis de la tension V_O à la sortie d'un circuit

- Le fabricant garantit
  - la durée max. garantie d'un flanc montant en sortie : t_{TLH} . Exemple : quelques ns
  - la durée maximum garantie d'un flanc descendant en sortie : t_{THL}

18.10.3 Conclusion : Peut-on connecter une sortie à une entrée ?

⇒ On peut connecter la sortie d'un circuit (dont les commutations durent max. t_{TLH} et t_{THL}) à l'entrée d'un autre circuit (qui tolère des commutations durant max. t_r et t_f)
si $t_{TLH} \leq t_r$ et $t_{THL} \leq t_f$

18.11 EXERCICES

67. Ces circuits sont-ils compatibles au point de vue statique ? Justifiez.
    - Circuit 1 : $V_{IH} = 3.15V$, $V_{IL} = 1.35V$, $V_{OH} = 4.4V$, $V_{OL} = 0.1V$
    - Circuit 2 : $V_{IH} = 1.5V$, $V_{IL} = 0.5V$, $V_{OH} = 1.9V$, $V_{OL} = 0.1V$

68. Téléchargez la datasheet d'un des fabricants et dessinez le graphique des plages de tensions du composant suivant : 74LS08 (alimenté en 5V)
69. Même exercice pour le 74HCT00 (alimenté en 5V)
70. Même exercice pour le 74HC32 (alimenté en 5V)

71. Au point de vue statique, peut-on connecter une sortie d'un 74LS08 (alimenté en 5V) à une entrée d'un 74HCT00 (alimenté en 5V) ?

72. Au point de vue statique, peut-on connecter une sortie d'un 74LS08 (alimenté en 5V) à une entrée d'un 74HC00 (alimenté en 5V) ?

73. Quelle sont les durées garanties des flancs de commutation du 74LS08 en sortie ; quelles sont les durées requises en entrées ?

    Au point de vue dynamique, peut-on connecter la sortie d'un 74LS08 à l'entrée d'un 74LS08 ?

19 Annexe : fonctions logiques élémentaires en DTL et RTL

- Note : RTL = Resistor Transistor Logic, DTL = Diode Transistor Logic

19.1 Réalisation de la fonction AND

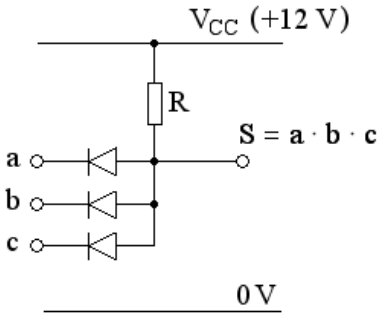


Figure 91 : Circuit AND à 3 entrées, en DL (il y a une diode par entrée)

- **Si on applique Vcc (niveau logique 1) aux 3 entrées a, b et c**
  - ⇒ les 3 diodes sont bloquées
  - ⇒ aucun courant ne passe dans R ⇒ la sortie est à Vcc, c.-à-d. à l'état logique 1
- Si on applique 0V (niveau logique 0) à une seule entrée, par exemple l'entrée a
  - ⇒ la diode correspondante est passante
  - ⇒ il apparaît entre ses bornes une d.d.p. d'environ 0,7V
  - ⇒ le potentiel de la sortie sera d'environ 0,7V

  De même si plusieurs entrées sont à 0 : le potentiel de la sortie est environ 0,7V ≅ 0V c.-à-d. l'état logique 0

Conclusion : ce circuit réalise bien la fonction AND .

19.2 Réalisation de la fonction OR

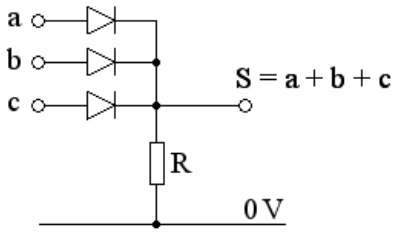


Figure 92 : Circuit OR à 3 variables, en logique DL (il y a une diode par entrée)

- **Si les 3 entrées a, b et c sont toutes au 0 logique** (les 3 tensions d'entrées à 0V)
  - ⇒ les 3 diodes sont bloquées ⇒ le courant dans R est nul
  - ⇒ la chute de tension aux bornes de R est nulle ⇒ la tension à la sortie est 0V
- **Si une seule entrée est à 1 ,** par ex. si $V_a = V_{CC}$ (+12V)
  - ⇒ la diode correspondante est passante
  - ⇒ il apparaît une d.d.p. d'environ 0,7V entre les bornes de cette diode
  - ⇒ le potentiel de la sortie est environ 12V – 0.7V = 11.3V (pour V_{CC} = 12V)

  De même si plusieurs entrées sont à 1 : le potentiel de la sortie est environ 11.3V ≅ V_{CC}

Conclusion : ce circuit réalise la fonction OR .

19.3 Réalisation de la fonction NOT (= "non" = inversion) en RTL

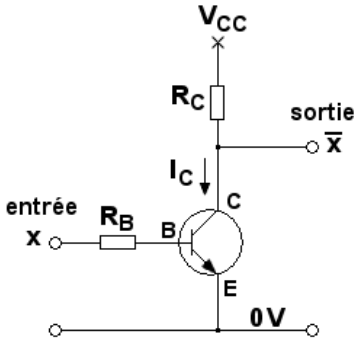


Figure 93 : Fonction NOT matérialisée en RTL

Note : on choisit les valeurs de R_B et R_C en sorte qu'à l'état passant, le transistor soit saturé (voir cours d'électronique analogique).

Dans un circuit logique,

- un transistor peut être considéré comme un simple interrupteur reliant les bornes C et E
- la tension V_{BE} commande la position de cet interrupteur :
  - Si $V_{BE} \geq 0.7\,V$, le transistor est "passant" : l'interrupteur est fermé :
    - un courant entre dans le transistor par la borne C et en sort par la borne E
    - V_{CE} est (quasi) nul
  - Si V_{BE} est clairement inférieur à 0,7V, le transistor est "bloqué" : l'interrupteur est ouvert :
    - le courant I_C est nul

Dès lors :

- Si on applique 0V (c.-à-d. un état logique "0") à l'entrée "x" :
  - $\Rightarrow$ le transistor est bloqué : il se comporte comme un interrupteur ouvert
  - $\Rightarrow$ le courant dans R_C est nul
  - $\Rightarrow$ la tension de sortie est donc égale à la tension d'alimentation (par ex. +12V) .

  Bref si on applique un "0" à l'entrée, on obtient un 1 à la sortie

- Si on applique V_{CC} (par ex. +12V), (c.-à-d. un état logique "1") à l'entrée "x" :
  - $\Rightarrow$ le transistor est passant : il se comporte comme un interrupteur fermé
  - $\Rightarrow$ le potentiel de sa borne C est quasi nul.

  Bref si on applique un "1" à l'entrée, on a donc bien un 0 à la sortie

- Conclusion : la sortie $s = \overline{x}$

19.4 Réalisation de la fonction NOR en DTL

On peut placer en cascade un circuit OR suivi d'un NOT pour former un circuit NOR.

Exemple avec 3 entrées :

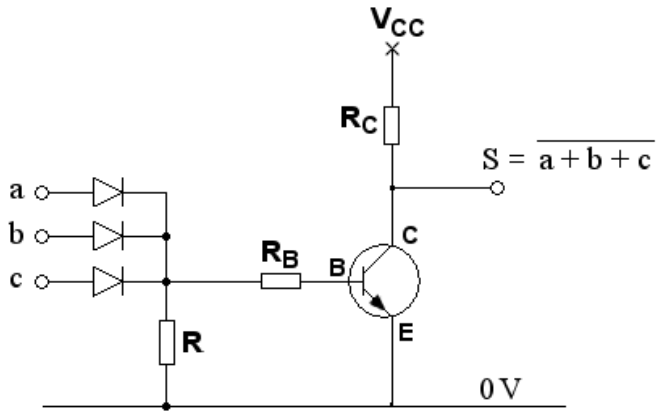


Figure 94 : Fonction NOR à 3 entrées en DTL

20 Annexe : Méthode de Karnaugh - Compléments

20.1.1 Table de Karnaugh à 5 variables et plus

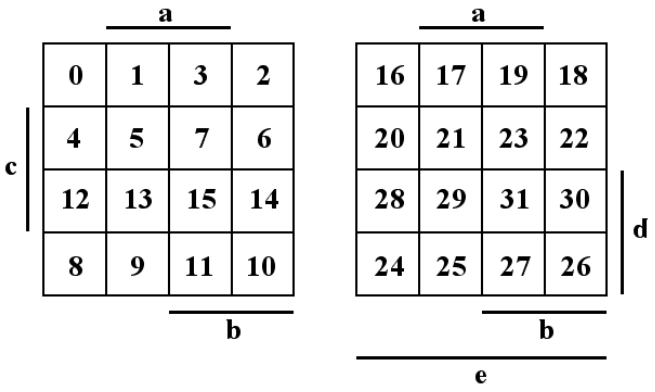


Figure 95 : Table de Karnaugh à 5 variables = 2 tables à 4 variables

- À 5 variables, pour garder la connexité visible,
  il faudrait dessiner la table de Karnaugh en 3 dimensions …
  il est plus facile de dessiner (**Figure 95**) 2 tables chacune de 4 variables (16 cases),
  mais évidemment, identifier les pavés (3-D) faits de cases connexes est alors plus difficile.

- À 6 variables, on peut de même dessiner 4 tables de 16 cases :

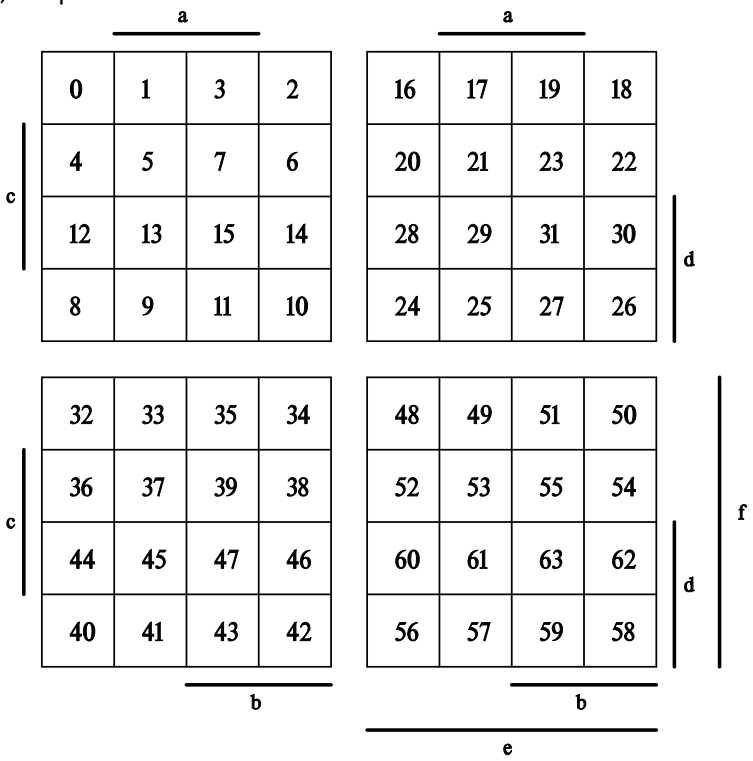


Figure 96 : Table de Karnaugh à 6 variables

- Mais peut-on réellement espérer identifier les pavés 4-D les plus grands possibles ?
  Il semble raisonnable de n'employer la méthode de Karnaugh que jusque 4 variables,
  et à partir de 5 variables, d'utiliser des outils informatiques.

20.1.2 Note : autres numérotations des cases d'une Table de Karnaugh

- Une table dont la numérotation est naturelle ne possède pas la propriété de connexité.

- Nous avons adopté une certaine numérotation des Tables de Karnaugh,
  mais il existe d'autres ordres des cases qui assurent la connexité
  (ils reviennent à intervertir lignes et colonnes, et/ou certaines lignes entre elles,
  et/ou certaines colonnes entre elles).
  Peu importe du moment que la table est connexe, *et que vous montrez clairement vos conventions*.

- La numérotation des états d'entrée dépend évidemment de la **pondération qu'on a choisi pour les entrées** (à 4 variables : d, c, b et a nous avons choisi d = poids fort, et a = poids faible, ceci dans la Table de vérité et dans la table de Karnaugh)

20.1.3 Couverture des aléas d'une fonction combinatoire

- Reprenons la fonction d'un mux 2 $\rightarrow$ 1 : f (s, a, b) = $\overline{s}$. a + s. b
  - si s = 0, le mux sélectionne l'entrée a : il copie la valeur de a sur la sortie f
  - si s = 1, le mux sélectionne l'entrée b : il copie la valeur de b sur la sortie f
- Considérons la fonction g (s, a, b) = $\overline{s}$. a + s . b + a .b
- Le théorème du consensus énonce que : f = g
  Nous avons démontré ce théorème en écrivant la table de vérité des 2 fonctions f et g
- Voici, dans la table de Karnaugh, les pavés correspondant aux termes de ces 2 fonctions :

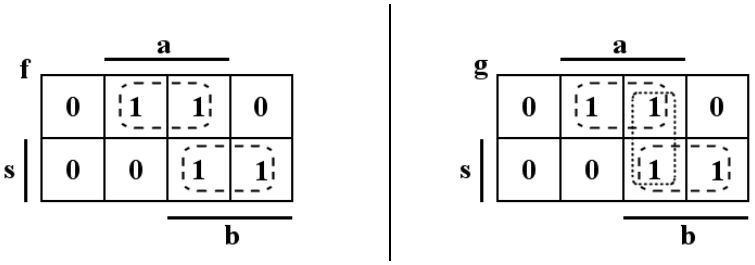


Figure 97 : Table de Karnaugh du mux 2 $\rightarrow$ 1, fonctions f et g

- Dans la fonction g, le pavé de couverture a . b couvre deux "1" déjà couverts dans f ;
  ce pavé de couverture assure, dans le cas a = b = 1 , que
  g reste à 1 indépendamment de la valeur de s (et de son changement éventuel),
  puisque le monôme a . b ne contient pas la variable s

- En généralisant cet exemple :
  on *couvre un aléa* en ajoutant un pavé qui couvre deux "1" contigus
  qui étaient au départ couverts par des pavés différents.

  L'aléa couvert (le risque éliminé) est celui causé
  par la variable d'entrée qui change lorsqu'on change de case dans le pavé.

21 Annexe : Réalisation d'une fonction par mux ou démux

- Réaliser une table de vérité donnée, à l'aide de multiplexeurs et/ou de démultiplexeurs est très facile. Hélas, les circuits obtenus ont un câblage assez lourd et encombrants.

  A titre d'information, voici comment procéder.

21.1 Réalisation par mux

- Un mux à n variables de commandes (et 2^n entrées) peut être utilisé comme « circuit générateur universel d'une fonction combinatoire de n variables » : pour cela, il suffit de :
  - piloter les n entrées de commande (sélection) du mux par les n variables d'entrées de la fonction
  - et positionner les 2^n entrées du mux aux valeurs de sortie spécifiées dans la table de vérité de la fonction. (c.-à-d. : à l'entrée k du mux on applique la valeur de sortie spécifiée dans la ligne k de la table de vérité).

- Exemple : on demande de réaliser la fonction $s = XOR(a,b,c)$ à l'aide d'un mux
  Cette fonction a 3 variables d'entrée $\Rightarrow$ il faut un mux à 3 entrées de commande (sélection), et $2^3 = 8$ entrées

État	Entrées			Sortie
n°	c	b	a	$a \oplus b \oplus c$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Tableau 65 : Table de définition de la fonction

$\Rightarrow$

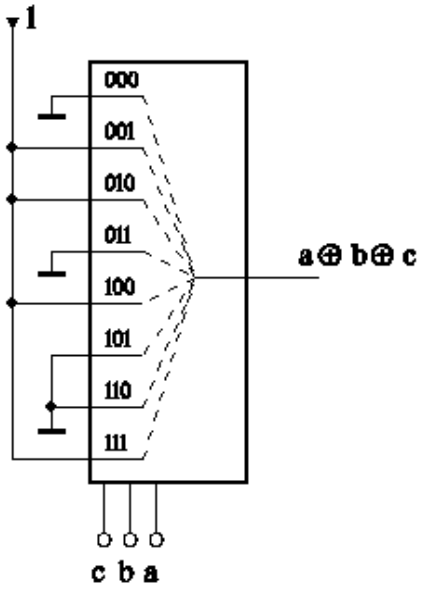


Figure 98 : Réalisation d'une fonction au moyen d'un mux

21.2 Réalisation par démux, et une porte (OR, NAND, NOR, NAND)

- Cette méthode est parfois employée pour réaliser plusieurs fonctions dépendant des mêmes variables :
  - on a besoin d'un seul démux pour l'ensemble des fonctions,
  - et d'une porte (OR, NAND, NOR ou NAND) pour chaque fonction.

- Exemple: pour réaliser les 2 fonctions des mêmes 3 variables :
  $s = a \oplus b \oplus c$ et $Maj(a,b,c)$:

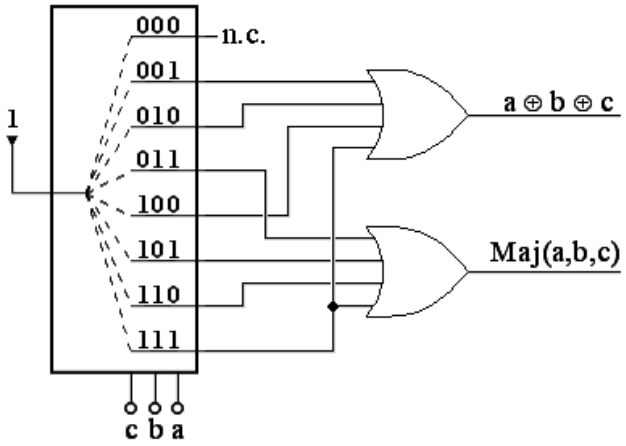


Figure 99 : Démux type 74xx238 à sorties actives hautes

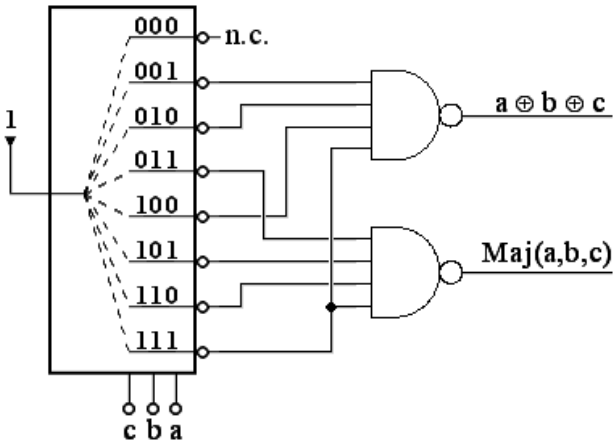


Figure 100 : Démux type 74xx138 à sorties actives basses

(transformation de la **Figure 99** par De Morgan)

- À un moment donné, une seule sortie est à 1
- On entre dans un OR les sorties du démux correspondant aux minterms à "1" (on fait une SDP)
- Le OR doit avoir autant d'entrées qu'il y a de "1" dans la fonction à réaliser.

- Si une fonction « s » comporte plus de "0" que de "1", il est plus économique de connecter les sorties du démux pour lesquelles la fonction s vaut "0", à un NOR ou à un AND :

Exemple :

État n°	Entrées c b a	Sortie s
0	0 0 0	1
1	0 0 1	1
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	1
7	1 1 1	1

Tableau 66 : Table de définition

⇒

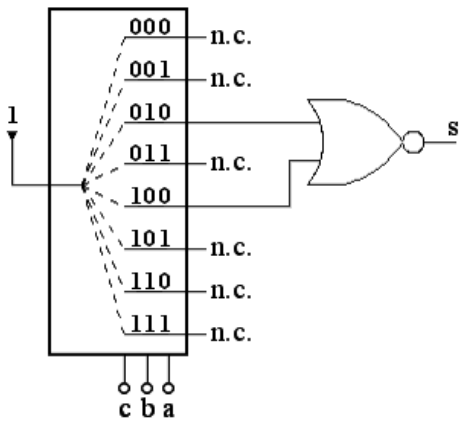


Figure 101 : Démux type 74xx238 (une seule sortie est à "1")

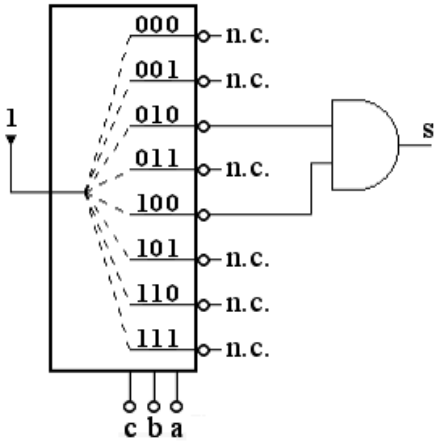


Figure 102 : Démux type 74xx138 une seule sortie est à "0"

(transformation de la **Figure 101** par De Morgan)

(la sortie vaut 0 si l'état vaut 2 ou 4).

22 Annexe : PDS standard d'une fonction combinatoire

22.1 PDS standard d'une fonction combinatoire

- Un maxterm est un OR de toutes les variables d'entrées, chaque variable d'entrée pouvant y apparaître sous forme vraie, ou sous forme complémentée

- Un système à n entrées a donc 2^n maxterms, notés M_i .
  Chaque maxterm vaut 0 pour un et un seul état d'entrée. On en déduit :

> Toute fonction combinatoire est égale au produit logique (AND) des maxterms associés aux états d'entrée pour lesquels la fonction vaut 0

  Cette expression est appelée "PDS standard" de la fonction [1] .

- Exemple : soit la fonction s de 2 variables b a. On demande le PDS standard de s

n°	entrées b	entrées a	sortie s		maxterms $M_0 = b+a$	$M_1 = b+\bar{a}$	$M_2 = \bar{b}+a$	$M_3 = \bar{b}+\bar{a}$
0	0	0	1		0	1	1	1
1	0	1	1		1	0	1	1
2	1	0	0	←	1	1	0	1
3	1	1	0	←	1	1	1	0

Tableau 67 : Recherche de la forme conjonctive d'une fonction s
A droite de la table de vérité, on a ajouté les maxterms

  Solution :

  - On recherche les 0 dans la colonne s
  - D'où : PDS standard : $s(a, b) = M_2 . M_3 \Rightarrow s = (\bar{b}+a) . (\bar{b}+\bar{a})$

- Notation abrégée de la forme PDS standard :
  on écrit seulement les numéros des maxterms pour lesquels la fonction vaut 0 :
  exemple : au lieu d'écrire $F = M_0 . M_1 . M_3 . M_4 . M_7$, on écrit $F = \prod 0, 1, 3, 4, 7$

[1] "PDS" = Produit de Sommes; "standard", parce que les termes contiennent toutes les variables.
On dit aussi : "forme canonique en Produit de Sommes", ou "forme canonique conjonctive"

23 Annexe : Fonctions min et max d'une fonction incomplètement spécifiée

- On définit la **fonction minimale** d'une fonction incomplètement définie comme étant la fonction obtenue en remplaçant les "×" par des "0" .
  Parmi toutes les fonctions possibles, c'est celle qui comporte le minimum de "1" .

- On définit la **fonction maximale** d'une fonction incomplètement définie la fonction obtenue en remplaçant les "×" par des "1" .
  Parmi toutes les fonctions possibles, c'est celle qui comporte le maximum de "1" .

- Exemple : soit table de vérité suivante :

État	Entrées			Sortie
n°	c	b	a	F
0	0	0	0	0
1	0	0	1	×
2	0	1	0	1
3	0	1	1	1
4	1	0	0	×
5	1	0	1	0
6	1	1	0	1
7	1	1	1	×

Tableau 68 : Table de vérité

La fonction minimale est :

État	Entrées			Sortie
n°	c	b	a	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	**1**
3	0	1	1	**1**
4	1	0	0	0
5	1	0	1	0
6	1	1	0	**1**
7	1	1	1	0

Tableau 69 : fonction minimale

24 Annexe : équations des sorties des décodeurs, démux et mux

- Décodeur $n \rightarrow 2^n$: équations des sorties (dans le cas de sorties actives hautes)

  $s_i = K_i$ où les K_i sont les minterms formés à partir des n bits d'adresse

- Démux $1 \rightarrow n$: équations des sorties [(1)] :

  $s_i = K_i \cdot e$ où les K_i sont les minterms formés à partir des n bits d'adresse

- Mux $n \rightarrow 1$: équations de la sortie :

  $s = e_0 \cdot K_0 + e_1 \cdot K_1 + \ldots + e_{n-1} \cdot K_{n-1}$

  où les K_i sont les minterms formés à partir des n bits d'adresse

[(1)] Il s'agit du Démux dont la sortie est active haute (comme par exemple le Démux 74xx238).

25 Annexe : Synthèse par simplification algébrique "libre"

- La simplification algébrique "libre", consiste en l'application des lois de l'algèbre de Boole, à l'intuition, sans méthode systématique.

- L'idée de base est d'utiliser la relation du " tiers exclus" :

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

En effet, l'application systématique de cette relation permet de réduire progressivement une somme de produit, comme le montre l'exemple type ci-dessous :

$$b \cdot a + b \cdot \overline{a} = b \cdot (a + \overline{a}) = b \cdot 1 = b$$

Hélas, sans procédure systématique pour nous guider, on "rate" des regroupements possibles, car il faut souvent dédoubler des termes afin de pouvoir effectuer les simplifications :

- Exemple :

N°	b a	F = a + b
0	0 0	0
1	0 1	**1**
2	1 0	**1**
3	1 1	**1**

$K_0 = \overline{b} \cdot \overline{a}$	$K_1 = \overline{b} \cdot a$	$K_2 = b \cdot \overline{a}$	$K_3 = b \cdot a$
1	0	0	0
0	**1**	0	0
0	0	**1**	0
0	0	0	**1**

Tableau 70 : fonction OR, minterms

$F = K_1 + K_2 + K_3$	(SDP standard du OR : somme de minterms)
$= \overline{b} \cdot a + b \cdot \overline{a} + b \cdot a$	
$= \overline{b} \cdot a + b \cdot \overline{a} + b \cdot a + b \cdot a$	(on a dédoublé le dernier terme, mais comment peut-on deviner ici que cela va nous permettre une simplification plus loin !!!?)
$= \overline{b} \cdot a + b \cdot a + b \cdot \overline{a} + b \cdot a$	(on a déplacé un des 2 termes dédoublés)
$= (\overline{b} + b) \cdot a + b \cdot (\overline{a} + a)$	(mise en évidence de a, mise en évidence de b)
$= 1 \cdot a + b \cdot 1$	(relation du tiers exclus)
$= a + b$	(1 = neutre du ".")

Comme forme simplifiée, on retrouve bien un OR …

- Conclusion de cet exemple très simple (2 variables !) :
cette méthode n'offre aucune garantie et elle est risquée :
    - sans guide quelle chance a-t-on d'obtenir à un résultat optimal ?
    - avec de longues suites d'équations, le risque d'erreur humaine est *très élevé*.

26 Annexe : Codeur/décodeur Binaire pur ↔ Gray

- Rappel : pour coder un nombre en binaire, on utilise parfois le "code binaire réfléchi" ou "code Gray". Dans ce code, un seul bit change entre 2 codes successifs.

- Soit un entier $N \geq 0$,
  et soit G le code de Gray (aussi un entier ≥ 0) correspondant à N

  On démontre que $\boxed{G_{bin} = N_{bin} \oplus (N/2)_{bin}}$

  Exemples :

  $N = 8_{déc} \Rightarrow N/2 = 4_{déc} \Rightarrow G_{bin} = 1000_{bin} \oplus 0100_{bin} = 1100_{bin}$

  $N = 15_{déc} \Rightarrow N/2 = 7_{déc} \Rightarrow G_{bin} = 1111_{bin} \oplus 0111_{bin} = 1000_{bin}$

  $N = 67_{déc} \Rightarrow N/2 = 33_{déc} \Rightarrow G_{bin} = 100\,0011_{bin} \oplus 010\,0001_{bin} = 110\,0010_{bin}$

  Ou, en écrivant cette formule pour les bits de G_{bin} et N_{bin} :

  $$\boxed{G_i = N_i \oplus N_{i+1}}$$

  De là on déduit : $\boxed{N_{bin} = G_{bin} \oplus (N/2)_{bin}}$

  ou, en écrivant cette formule pour les bits de G_{bin} et N_{bin} :

  $$\boxed{N_i = G_i \oplus N_{i+1}}$$

  - (G_i , le bit de rang i en code Gray, est égal au XOR entre
    N_i , bit de rang i du code binaire naturel,
    et N_{i+1}, bit de rang i+1 du code binaire naturel ;
  - le dernier bit de gauche n'ayant pas de rang supérieur, on prend celui-ci égal à 0.

- De plus, partons de $G_i = B_i \oplus B_{i+1}$

  Ajoutons B_{i+1} aux 2 membres $\Rightarrow G_i \oplus B_{i+1} = B_i \oplus B_{i+1} \oplus B_{i+1}$,

  Comme $a \oplus a = 0$ $\Rightarrow G_i \oplus B_{i+1} = B_i$

  $\Rightarrow \boxed{B_i = G_i \oplus B_{i+1}}$

26.1 Circuit Codeur-Décodeur Binaire naturel ↔ Gray

- On peut réaliser un seul circuit pour la conversion d'un code vers l'autre et inversement. Pour cela, on introduit une variable de commande (mode) M : on convient que :
  - si M = 0 : on calcule le code Gray à partir du binaire naturel : $G_i = B_i \oplus B_{i+1}$
    Les entrées sont les B_i, et les sorties sont les G_i
  - si M = 1 : on calcule le code binaire naturel : $B_i = G_i \oplus B_{i+1}$
    Les entrées sont les G_i, et les sorties sont les B_i

- On peut donc réécrire les relations ci-dessus, en fonction des entrées E_i et des sorties S_i :
  - si M = 0 : $S_i = E_i \oplus E_{i+1}$
  - si M = 1 : $S_i = E_i \oplus S_{i+1}$

  ⇒ Pour coupler les 2 fonctionnements, il suffit donc, dans l'étage i,
  de multiplexer soit l'entrée suivante E_{i+1} soit la sortie suivante S_{i+1}
  (le choix étant contrôlé par l'entrée M),
  et d'envoyer la sortie de ce mux, et l'entrée E_i, dans un XOR :

$$S_i = E_i \oplus (\overline{M} \cdot E_{i+1} + M \cdot S_{i+1})$$

- D'où le schéma :

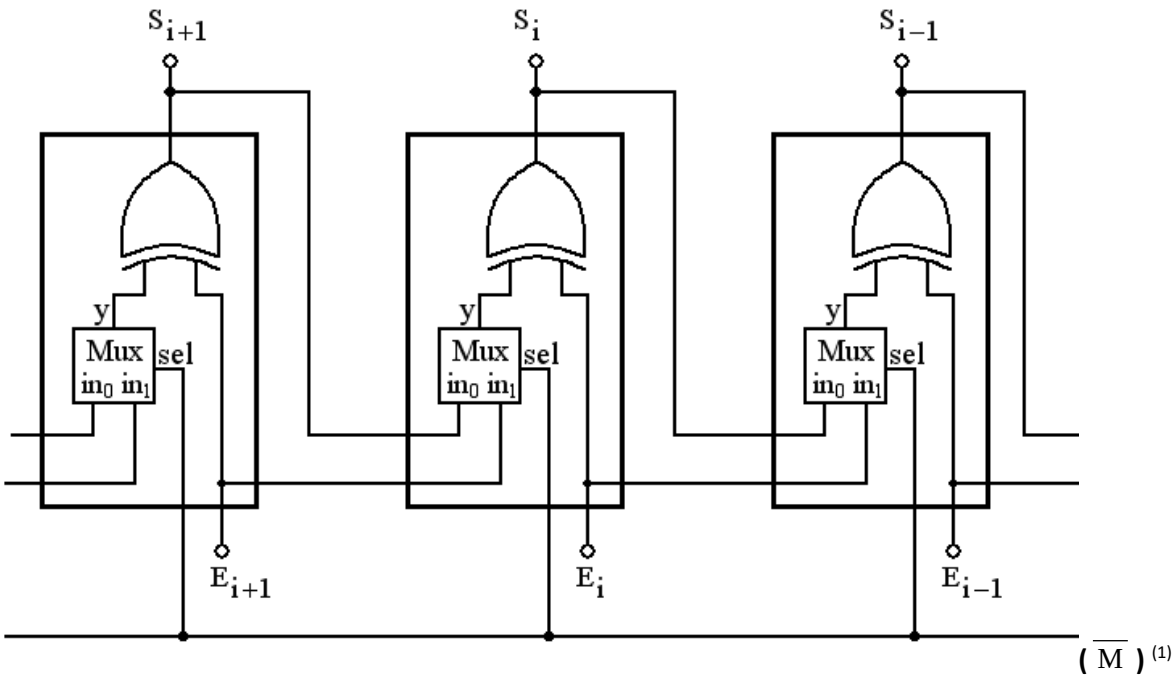


Figure 103 : Codeur Gray ↔ Nombre binaire pur, structure itérative (poids forts côté gauche)

$$S_i = E_i \oplus (\overline{M} \cdot E_{i+1} + M \cdot S_{i+1})$$

- On obtient bien une structure itérative, avec
  - 2 signaux interstitiels d'entrée : E_{i+1} , S_{i+1} (venant de la cellule de gauche)

    La cellule de poids le plus fort (celle la plus à gauche) n'ayant pas de voisine de gauche, on met à 0 ses 2 entrées interstitielles (2)
  - 2 signaux interstitiels de sortie : E_i , S_i (allant vers la cellule de gauche)
  - une entrée parallèle E_i
  - une sortie parallèle S_i .
  - Une cellule contient un XOR et un mux $2 \rightarrow 1$

(1) On peut aussi inverser les connexions des entrées in0 et in1 si l'on souhaite utiliser « M » pour le bit de sélection plutôt que son complément. Autre façon : on utilise « M » pour la sélection, on maintient telles quelles les connexions de in0 et in1 mais on inverse la convention sur l'état du bit de sélection ; il faut alors adapter les expressions logiques.

(2) La cellule la plus à gauche revient alors à la fonction identité, mais on utilise quand même une cellule complète pour permettre une extension à un nombre plus élevé de bits.

27 Annexe : Code ASCII étendu (8 bits)

- Rappel : On appelle "caractère" un groupe de bits (dans les cas simples de 6 à 8 bits), dont les différentes combinaisons représentent les lettres majuscules, les minuscules, les chiffres décimaux, les signes de ponctuation et des codes de commande (par exemple : retour en début de ligne).

Voici le tableau du code ASCII étendu en 8 bits, tel qu'il est utilisé sur les PC :

N° déc	Car	N° déc	Car	N° déc	Car	N° déc	Car	N° déc	Car	N° déc	Car	N° déc	Car	N° déc	Car
000	nul	032	esp	064	@	096	`	128	Ç	160	ááá	192		224	a
001	☺	033	!	065	A	097	a	129	ü	161	í	193	−	225	ß
002	☹	034	"	066	B	098	b	130	é	162	ó	194	−	226	G
003	♥	035	#	067	C	099	c	131	â	163	ú	195	+	227	p
004	♦	036	$	068	D	100	d	132	ä	164	ñ	196	−	228	S
005	♣	037	%	069	E	101	e	133	à	165	Ñ	197	+	229	s
006	♠	038	&	070	F	102	f	134	å	166	ª	198	¦	230	µ
007	bip	039	'	071	G	103	g	135	ç	167	º	199	¦	231	t
008	◘	040	(	072	H	104	h	136	ê	168	¿	200	+	232	F
009	tab	041	)	073	I	105	i	137	ë	169	¬	201	+	233	T
010	LF	042	*	074	J	106	j	138	è	170	¬	202	−	234	O
011	Home	043	+	075	K	107	k	139	ï	171	½	203	−	235	d
012	FF	044	,	076	L	108	l	140	î	172	¼	204	¦	236	8
013	CR	045	−	077	M	109	m	141	ì	173	¡	205	−	237	f
014	♫	046	.	078	N	110	n	142	Ä	174	«	206	+	238	e
015	☼	047	/	079	O	111	o	143	Å	175	»	207	−	239	n
016	➔	048	0	080	P	112	p	144	É	176	¦	208	−	240	=
017	➔	049	1	081	Q	113	q	145	æ	177	¦	209	−	241	±
018	↕	050	2	082	R	114	r	146	Æ	178	¦	210	−	242	=
019	!!	051	3	083	S	115	s	147	ô	179	¦	211	+	243	=
020	∏	052	4	084	T	116	t	148	ö	180	¦	212	+	244	(
021	§	053	5	085	U	117	u	149	ò	181	¦	213	+	245	)
022	≈	054	6	086	V	118	v	150	û	182	¦	214	+	246	÷
023	□	055	7	087	W	119	w	151	ù	183	+	215	+	247	~
024	↑	056	8	088	X	120	x	152	ÿ	184	+	216	+	248	°
025	↓	057	9	089	Y	121	y	153	Ö	185	¦	217	+	249	·
026	→	058	:	090	Z	122	z	154	Ü	186	¦	218	+	250	·
027	←	059	;	091	[	123	{	155	¢	187	+	219	¦	251	v
028	CuR	060	<	092	\	124	|	156	£	188	+	220	_	252	n
029	CuL	061	=	093	]	125	}	157	¥	189	+	221	¦	253	²
030	CuU	062	>	094	^	126	~	158	P	190	+	222	¦	254	¦
031	CuD	063	?	095	_	127	□	159	ƒ	191	+	223	¯	255	

Tableau 71 : Codage ASCII 8 bits, utilisé par les PC

28 Annexe : Une brève histoire de l'électronique numérique

- *« Tous les hommes sont mortels. Tous les Grecs sont des hommes. Donc tous les Grecs sont mortels »* est un raisonnement logique, écrit en phrases (voir Wikipédia, logique **aristotélicienne**)
- Milieu du 19e siècle : **George Boole** montre qu'il est possible de prouver un résultat logique au moyen d'opérations très simples (addition et mutiplication) sur des nombres binaires.
  A l'époque, la technologie ne permet pas de réaliser une application concrète.
- 1906 : **Lee de Forest** invente le premier tube à vide servant d'amplificateur : la triode.
- 1918 : **William Eccles** and **Frank Jordan** inventent le premier circuit électronique de mémoire binaire, sur base de deux amplificateurs à tube à vide.
- 1930 : **Claude Shannon** montre que la logique « booléenne » est l'outil idéal pour décrire des circuits logiques et combiner des signaux à valeurs binaires.
- 1945 : le mathématicien **John von Neumann** propose la structure actuelle des ordinateurs
- 1946 : **John Mauchly** et **John Eckert** mettent en service l'ENIAC
  (Electronic Numerical Integrator and Computer)
  L'ENIAC est un des premier ordinateurs électronique universel de grande puissance.
  Il contient 17 468 tubes à vide, et sa puissance consommée est de 174 kW (kilowatts !) .

  Sans compter l'alimentation et le conditionnement d'air,
  il occupe 42 armoires électriques, il mesure 6 m de haut, 27 m de long, et pèse 30 tonnes !

  Il fait les calculs sur des nombres décimaux (10 chiffres décimaux + signe) :
  - l'addition dure 0,2 ms (deux dixièmes de milliseconde),
  - la multiplication 2,8 ms ; la division dure 24 ms

  Il est très peu fiable :
  - MTBF (Mean time between failure): 5 h 36 min
  - Operating ratio : sa disponibilité est de 70 %
    (pendant 30% du temps il est est à l'entretien ou en réparation)
- 1947 : **William Shockley** invente le transistor.
  Le transistor est fabriqué à partir du matériau qui, avec l'oxygène, est le plus répandu sur Terre : le silicium.
- En 1958, **Jack Kilby** (**Texas Instruments,** Dallas, USA) invente le circuit intégré.
- 1971 : Premier "microprocesseur" : l' **Intel 4004 est un** CPU (Central Processing Unit) 4 bits, réalisé sur une seule puce.
- 1972 : Premier microprocesseur 8 bits : l' **Intel 8008**
- 1974 : Microprocesseur performant, facile à utiliser, bon marché : L**'Intel 8080**
  Le 8080 a lancé l'utilisation des microprocesseurs pour de nombreuses applications industrielles, ainsi que pour les (précurseurs des) ordinateurs personnels
- 2016 : un ordinateur portable à 8 CPUs 64 bits coûte moins de 1000 euros taxes comprises

29 Bibliographie

- « Systèmes numériques » , Thomas L FLOYD, 9e éd. (2006) avec CD, 885 pages,
  ISBN : 978-2-89377-324-7, http://www.goulet.ca/
  Ceci est une traduction en français (coût environ 63 €) ; l'original anglais est meilleur marché

- « Guide du technicien en électronique » , C. Cimelli, R. Bourgeron,
  Édition 2007, 304 pages, ISBN 978-2-01-180441-9, http://www.eyrolles.com/
  (coût environ 27 €)

- « Bebop to the Boolean Boogle. An Unconventional Guide to Electronics » ,
  Clive "Max" Maxfield, 3rd ed. (2009) Newnes - Elsevier http://www.elsevier.com
  ISBN: 978-1-85617-507-4
  (coût environ 31 €)

- http://www.nxp.com

- http://www.ti.com

- http://www.onsemi.com

- http://www.fairchildsemi.com

- http://www.alldatasheet.com