UE Electrical Engineering
2025

UE Electrical Engineering : électronique numérique

Polycopié de cours 1/2
Logique combinatoire et circuits MOS

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Sommaire

Chapitre 1 : Représentation de l'information numérique et arithmétique binaire.....1

1. INTRODUCTION.....1

2. REPRESENTATION NUMERIQUE DE L'INFORMATION 1

2.1 REPRESENTATION POLYNOMIALE D'UN NOMBRE 1
2.2 BASES DE NUMERATION USUELLES ET REPRESENTATION DES NOMBRES POSITIFS CONVERSIONS ENTRE BASES DE NUMERATION 2
2.2.1 Base b vers base 10 2
2.2.2 Base 10 vers *base b* 3
2.2.3 Base 2 vers base 2^n 6
2.2.4 Base 2^n *vers base 2* 7
2.3 REPRESENTATION BINAIRE DES NOMBRES SIGNES 7
2.3.1 Représentation en complément à 2 7
2.3.2 Représentation module + signe 9
2.3.3 Représentation binaire décalée 10
2.4 REPRESENTATION DES NOMBRES FRACTIONNAIRES 10
2.4.1 Codage en virgule fixe 10
2.4.2 Codage en virgule flottante 11
2.5 CLASSIFICATION DES CODES BINAIRES 12
2.5.1 Codes pondérés 12
2.5.1.1 Le code binaire pur et ses dérivés (octal, hexadécimal) 12
2.5.1.2 Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal) 13
2.5.2 Codes non pondérés 13
2.5.2.1 Code excédent 3 ou excess 3 13
2.5.2.2 Code binaire réfléchi ou code de Gray 14
2.5.2.3 Codes redondants 14
2.5.3 Codes alphanumériques 15

3. OPERATIONS ARITHMETIQUES.....16

3.1 ADDITION ET SOUSTRACTION 16
3.2 MULTIPLICATION ET DIVISION 16
3.2.1 Multiplication et division par 2^k 16
3.2.2 Cas général 17

4. CONCLUSION 18

5. BIBLIOGRAPHIE.....18

Chapitre 2 : Propriétés des variables et fonctions logiques 19

1. INTRODUCTION.....19

2. PROPRIETES DE L'ALGEBRE DE BOOLE 19

2.1 DEFINITIONS 19
2.2 TABLE DE VERITE D'UNE FONCTION LOGIQUE 19
2.3 LES FONCTIONS LOGIQUES ELEMENTAIRES 20
2.3.1 La fonction de complémentation ou fonction NON 20
2.3.2 La fonction produit logique ou fonction ET 20
2.3.3 La fonction addition logique ou fonction OU 20
2.3.4 Propriétés des fonctions NON, ET, et OU 21
2.3.5 Opérateurs secondaires 22
2.3.5.1 La fonction NON ET ou NAND : $\overline{A.B}$ 22
2.3.5.2 La fonction NON OU ou NOR : $\overline{A+B}$ 23
2.3.5.3 Quelques propriétés des fonctions NON ET et NON OU 23
2.3.5.4 La fonction OU exclusif (abrégé OUEX ou XOR) : $A \oplus B = A.\overline{B} + \overline{A}.B$ 23
2.3.5.5 La fonction ET inclusif (abrégé XNOR) : $\overline{A \oplus B} = A.B + \overline{A}.\overline{B}$ 24
2.3.6 Opérateurs complets 25

3. REPRESENTATION DES FONCTIONS LOGIQUES 25

3.1 FORMES ALGEBRIQUES DISJONCTIVES, CONJONCTIVES, CANONIQUES 25
3.2 REPRESENTATIONS DE REFERENCE D'UNE FONCTION LOGIQUE 26
3.3 CRITERES DE CHOIX D'UNE REPRESENTATION 26

4. SIMPLIFICATION DES FONCTIONS LOGIQUES 28

4.1 POURQUOI SIMPLIFIER LES FONCTIONS LOGIQUES ? 28
4.2 SIMPLIFICATION ALGEBRIQUE 28
4.2.1 Exemple 1 : simplification de $F_1 = BC + AC + AB + B$. 28
4.2.2 Exemple 2 : simplification de $F_2 = (A + \overline{B})(A\overline{B} + C)C$ 28
4.2.3 Exemple 3 : simplification de $F_3 = \overline{A}\,B\overline{C} + AB\,\overline{C} + ABC + \overline{A}\,BC$. 28
4.2.4 Exemple 4 : simplification de $F_4 = \overline{A}\,B + AC + BC$ 29
4.2.5 Exemple 5 : simplification de $F_5 = (\overline{A} + B)(A + C)(B + C)$. 29
4.2.6 Exemple 6 : simplification de $F_6 = (A\overline{B} + \overline{A}\,B).(AB + \overline{A}\,\overline{B})$ 29
4.2.7 Conclusion 29
4.3 SIMPLIFICATION PAR DIAGRAMME DE KARNAUGH 30
4.3.1 Introduction 30
4.3.2 Adjacence logique 30
4.3.3 Construction d'un diagramme de Karnaugh 30
4.3.3.1 Fonction de 2 variables 31
4.3.3.2 Fonction de 3 variables 31
4.3.3.3 Fonction de 4 variables 32
4.3.3.4 Fonctions de 5 et 6 variables 33
4.3.4 Principe de la simplification 34
4.3.4.1 Technique à appliquer sur un diagramme de Karnaugh quelconque 35
4.3.4.2 Exemples 36
4.3.4.3 Cas des fonctions incomplètement spécifiées 38
4.3.4.4 Les cas du OU exclusif et du ET inclusif 39
4.3.4.5 Conclusion 40

5. BIBLIOGRAPHIE 41

Chapitre 3 : Electronique des circuits logiques. Eléments de circuiterie MOS et CMOS 43

1. BREF HISTORIQUE DE L'ELECTRONIQUE ET EVOLUTION DES CIRCUITS INTEGRES.....43

1.1 LE TRANSISTOR BIPOLAIRE BJT (BIPOLAR JUNCTION TRANSISTOR).....43
1.2 LES PREMIERS CIRCUITS INTEGRES.....43
1.3 LE TRANSISTOR A EFFET DE CHAMP FET (FIELD EFFECT TRANSISTOR).....44
1.4 L'EVOLUTION DES CIRCUITS INTEGRES DEPUIS 1960.....44
1.4.1 Évolution de la densité d'intégration.....44
1.4.2 Évolution des circuits MOS.....46
1.4.3 Évolution des circuits bipolaires.....47
1.4.4 Les technologies alternatives.....47
1.4.5 Le marché des semi-conducteurs et des circuits intégrés.....48
1.4.6 Perspectives d'évolution des technologies sur silicium.....49

2. MODELE DU TRANSISTOR MOS UTILISE EN ELECTRONIQUE NUMERIQUE.....50

2.1 RAPPELS SUR LA STRUCTURE DU TRANSISTOR MOS.....50
2.2 ÉQUATIONS DE CONDUCTION DU TRANSISTOR MOS.....51
2.2.1 Transistor MOS canal N ou NMOS ($V_{DS} \geq 0$).....51
2.2.2 Transistor MOS canal P ou PMOS ($V_{DS} \leq 0$).....52
2.3 CAPACITES PARASITES DU TRANSISTOR MOS.....54
2.3.1 Capacité de grille.....54
2.3.2 Capacité des jonctions source/substrat C_{SB} et drain/substrat C_{DB}.....55
2.3.3 Valeurs numériques.....56

3. LES CIRCUITS LOGIQUES MOS.....57

3.1 ÉTUDE DE L'INVERSEUR NMOS A CHARGE RESISTIVE.....57
3.1.1 Comportement logique de l'inverseur.....57
3.1.2 Caractéristiques statiques de l'inverseur.....58
3.1.2.1 Niveaux de sortie.....58
3.1.2.2 Caractéristique de transfert en tension et marges de bruit.....60
3.1.2.3 Consommation statique.....62
3.1.2.4 Influence de la valeur de R sur les caractéristiques statiques de l'inverseur NMOS à charge résistive.....62
3.1.3 Caractéristiques dynamiques.....63
3.1.3.1 Temps de montée, de descente et temps de propagation.....63
3.1.3.2 Calcul des temps de montée et de descente pour l'inverseur NMOS à charge résistive. Résistance équivalente d'un transistor NMOS.....64
3.1.3.3 Consommation dynamique.....68
3.1.3.4 Influence de R sur les caractéristiques dynamiques de l'inverseur NMOS à charge résistive.....69
3.2 ÉTUDE DE L'INVERSEUR PMOS A CHARGE RESISTIVE.....70
3.3 LES INVERSEURS MOS REELS.....72
3.4 ETUDE DE L'INVERSEUR CMOS.....74
3.4.1 Comportement logique de l'inverseur.....74
3.4.2 Caractéristiques statiques de l'inverseur CMOS.....74
3.4.2.1 Tracé de la caractéristique de transfert.....74
3.4.2.2 Consommation statique.....79
3.4.3 Caractéristiques dynamiques de l'inverseur CMOS.....79
3.4.3.1 Consommation dynamique.....79
3.4.3.2 Temps de montée, de descente et temps de propagation.....79
3.5 CONSTRUCTION DE FONCTIONS COMBINATOIRES EN LOGIQUE CMOS.....82
3.5.1 Notation.....82
3.5.2 Opérateurs statiques.....82

3.5.2.1 Porte NON ET ou NAND 83
3.5.2.2 Porte NON OU ou NOR 84
3.5.2.3 Autres fonctions combinatoires 85
3.5.3 Opérateurs à base d'interrupteurs 87
3.5.3.1 L'interrupteur CMOS ou porte de transfert 87
3.5.3.2 Fonction OU exclusif 88
3.5.3.3 Opérateurs à sortie 3 états 89
3.6 PERFORMANCES STATIQUES ET DYNAMIQUES DES CIRCUITS LOGIQUES CMOS 90
3.6.1 Performances statiques 90
3.6.2 Performances dynamiques 90
3.6.2.1 Puissance dynamique 90
3.6.2.2 Temps de montée et de descente 90
3.6.2.3 Temps de propagation. Notions d'entrance et de sortance 91
3.6.2.4 Analyse de la capacité de charge C_L d'une porte logique 92
3.6.2.5 Capacité d'entrée minimale, entrance, et sortance 94
3.6.2.6 Temps de propagation d'un bloc logique et sortance 95
3.7 EVOLUTION DES PERFORMANCES DES CIRCUITS CMOS 96

4. BIBLIOGRAPHIE 98

Chapitre 4 : Fonctions et circuits combinatoires 99

1. DEFINITIONS 99

2. LES OPERATEURS DE TRANSCODAGE 99

2.1 DEFINITION 99
2.2 LES CODEURS 100
2.2.1 Exemples de codeurs 100
2.2.2 Codeur prioritaire 102
2.3 LES DECODEURS 102
2.3.1 Les principaux types de décodeurs 103
2.3.1.1 Les décodeurs binaires 103
2.3.1.2 Le décodeur BCD 104
2.3.2 Matrice de décodage 105
2.3.3 Association de décodeurs 106
2.3.4 Décodage à plusieurs niveaux 107
2.3.5 Applications des décodeurs 108
2.4 LES TRANSCODEURS 109
2.4.1 Exemple 1 : le transcodeur BCD/7 segments 109
2.4.2 Exemple 2 : les convertisseurs gray/binaire et binaire/Gray 110

3. LES OPERATEURS D'AIGUILLAGE 112

3.1 LES MULTIPLEXEURS 112
3.1.1 Exemples de multiplexeurs 113
3.1.2 Multiplexage de mots 114
3.1.3 Applications des multiplexeurs 115
3.2 LES DEMULTIPLEXEURS 116
3.2.1 Exemples de démultiplexeurs 116
3.2.2 Applications des démultiplexeurs 117

4. LES OPERATEURS DE COMPARAISON 118

4.1 DEFINITION 118
4.2 EXEMPLES DE COMPARATEURS 119

5. LES OPERATEURS ARITHMETIQUES 120

5.1 LES ADDITIONNEURS 120
5.1.1 Addition de deux bits 121
5.1.2 Addition de deux nombres binaires 122
5.1.2.1 Additionneur à retenue propagée (ripple carry adder) 122
5.1.2.2 Additionneur à retenue anticipée (carry look-ahead adder) 123
5.2 LES SOUSTRACTEURS 125
5.3 LES MULTIPLIEURS ET DIVISEURS 125
5.4 LES UNITES ARITHMETIQUES ET LOGIQUES 126
5.4.1 Exemple : le circuit intégré de référence xx382 126

6. BIBLIOGRAPHIE 129

Chapitre 1 : Représentation de l'information numérique et arithmétique binaire

1. Introduction

Les systèmes numériques complexes tels que les calculateurs doivent traiter des nombres, des chaînes de caractères alphanumériques, des instructions. A cette fin, ces informations sont représentées à l'aide de l'élément binaire, noté **eb** ou **bit** (**b**inary dig**it**). L'objectif de ce chapitre n'est pas de démontrer des théorèmes, ni de présenter de manière exhaustive « l'art de la numération », qui fait toujours l'objet de recherches, mais plutôt de rappeler les notions fondamentales du codage de l'information utilisé par les systèmes numériques.

2. Représentation numérique de l'information

Les informations numériques ou digitales sont des informations numérisées qui ne prennent en compte qu'un nombre fini de valeurs discrètes. Nous allons étudier ici les divers codages employés pour les traiter.

2.1 Représentation polynomiale d'un nombre

De manière générale tout nombre N exprimé dans une base b peut se décomposer sous la forme polynomiale suivante :

$$N_{(b)} = S(a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + \ldots + a_{-m} b^{-m}) \qquad \text{(équation 1)}$$

avec

- S est le **signe** du nombre
- a_i est le **symbole de rang *i***, $a_i \in \mathbf{N}$ et $0 \leq a_i < b$
- a_n est le **symbole de poids le plus fort** (MSB : Most Significant Bit si $b = 2$), et a_{-m} est le **symbole de poids le plus faible** (LSB : Least Significant Bit si $b = 2$)

Le nombre $N_{(b)}$ s'exprime en numérotation de position par $S\, a_n a_{n-1} \ldots a_0 \,,\, a_{-1} \ldots a_{-m}$. Les symboles $a_n a_{n-1} \ldots a_0$ et $a_{-1} \ldots a_{-m}$ représentent respectivement la **partie entière** et la **partie fractionnaire** de N.

On appelle **dynamique** ou **amplitude de codage** d'une représentation la différence entre le plus grand nombre et le plus petit nombre représentables. On appelle **résolution** ou **précision** d'une représentation la différence entre deux nombres consécutifs. A titre d'exemple pour une représentation décimale d'entiers positifs sur 5 chiffres, la dynamique est égale à 99999 et la résolution est égale à 1.

2.2 Bases de numération usuelles et représentation des nombres positifs. Conversions entre bases de numération

Les bases de numération les plus utilisées sont la base **décimale** ($b = 10$), la base **binaire** ($b = 2$), et les bases dérivées de la base binaire : base **octale** ($b = 8$) et base **hexadécimale** ($b = 16$).

La numération binaire utilise les 2 bits 0 et 1, la numération octale utilise 8 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, et la numération hexadécimale utilise 16 symboles : 0, 1, 2, ..., 9, A, B, C, D, E, F (les symboles A à F ont pour équivalents décimaux les nombres 10 à 15).

Le système binaire et ses dérivés sont ceux utilisés pour le codage des informations dans les systèmes numériques. La base 16 est couramment utilisée car elle peut être considérée comme une écriture condensée de l'écriture binaire et, par conséquent, sa conversion vers le binaire est particulièrement aisée.

Les conversions les plus utilisées sont les suivantes

- base b vers base 10
- base 10 vers base b
- base 2 vers base 2^n (8 ou 16)
- base 2^n (8 ou 16) vers base 2

2.2.1 Base b vers base 10

Pour convertir un nombre d'une base b vers la base décimale, on utilise la **méthode** dite **des additions** qui consiste à utiliser la représentation du nombre sous forme polynomiale (équation 1).

Exemple 1 : conversion du nombre binaire entier $N_{(2)} = 1101\ 0011_{(2)}$ en base 10.

$$N = 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = 128 + 64 + 16 + 2 + 1 = 211_{(10)}$$

Exemple 2 : conversion du nombre binaire fractionnaire $N_{(2)} = 110011{,}1001_{(2)}$ en base 10

$$N = 1.2^5 + 1.2^4 + 1.2^1 + 1.2^0 + 1.2^{-1} + 1.2^{-4} = 51{,}5625_{(10)}$$

Exemple 3 : conversion du nombre octal entier $N_{(8)} = 4513_{(8)}$ en base 10

$$N = 4.8^3 + 5.8^2 + 1.8^1 + 3.8^0 = 2379_{(10)}$$

Exemple 4 : conversion du nombre hexadécimal fractionnaire $N_{(16)} = 1B20,8_{(16)}$ en base 10

$$N = 1.16^3 + 11.16^2 + 2.16^1 + 8.16^{-1} = 6944,5_{(10)}$$

N. B. : La méthode des additions requiert la connaissance des puissances successives de la base de départ.

2.2.2 Base 10 vers base b

- **Nombres entiers**

Pour effectuer une conversion d'un entier décimal dans une autre base on applique la **méthode des divisions successives** : on effectue des divisions successives du nombre par cette base, les restes successifs forment alors le nombre converti.

A titre d'exemple, dans le cas d'une conversion d'un nombre décimal en son équivalent binaire, on réalisera des divisions successives par 2. Les restes de ces divisions formeront le nombre converti dans la base 2.

Exemple 1 : conversion de $N_{(10)} = 52$ en base 2

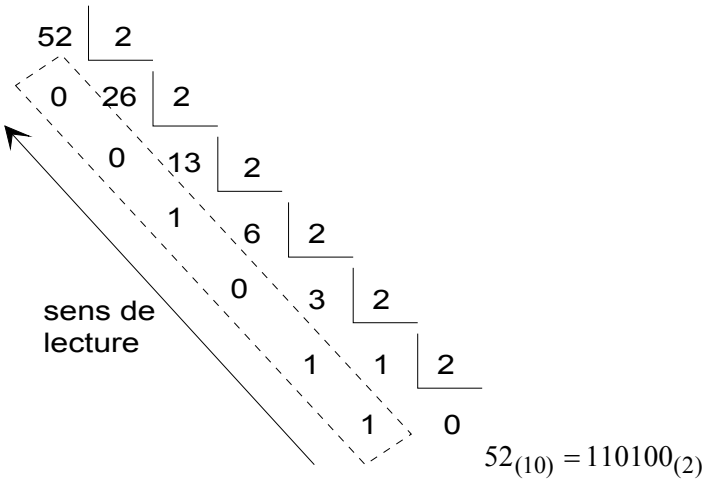


figure 1.1 : conversion de $52_{(10)}$ en base 2 par divisions successives par 2

Exemple 2 : conversion de $N_{(10)} = 90$ en base 8

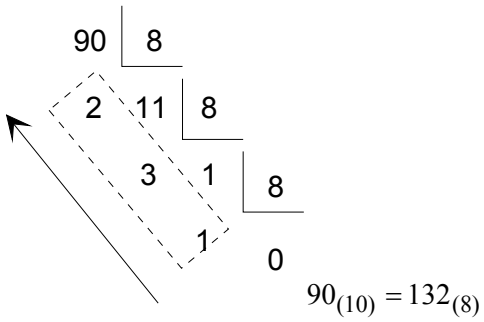


figure 1.2 : conversion de $90_{(10)}$ en base 8 par divisions successives par 8

Chaque division revient à opérer un décalage à droite d'une position et permet ainsi d'isoler un bit dans la partie fractionnaire.

Si les puissances successives de la base d'arrivée sont connues, on peut également, plutôt que d'utiliser la méthode précédente, effectuer la transformation par **soustractions successives** de ces puissances. Cette méthode est illustrée sur les deux exemples traités précédemment.

Exemple 1 : conversion de $N_{(10)} = 52$ en base 2

$32(=2^5) \le N < 64(=2^6)$, on peut donc retrancher 32 à N : $N = 32 + 20$,
$16(=2^4) \le 20 < 32(=2^5)$, on peut retrancher 16 : $N = 32 + 16 + 4$,
4 est une puissance de 2, l'itération est donc terminée. On en déduit
$N = 2^5 + 2^4 + 2^2 = 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 0.2^0 = 110100_{(2)}$.

Exemple 2 : conversion de $N_{(10)} = 90$ en base 8

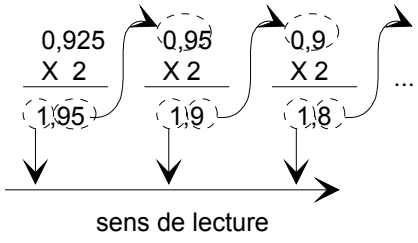
$64(=8^2) \le N < 512(=8^3)$, on retranche 64 à N : $N = 64 + 26$,
$8 \le 26 < 64(=8^2)$, on retranche 8 : $N = 64 + 8 + 18$,
on peut de nouveau retrancher 2 fois 8 à 18, on obtient alors : $N = 64 + 3.8 + 2$. Puisque 2 est inférieur à 8, l'itération est terminée, d'où $N = 1.8^2 + 3.8^1 + 2.8^0 = 132_{(8)}$.

- **Nombres fractionnaires**

Pour convertir un nombre fractionnaire de la base 10 vers une autre base, il faut procéder en deux étapes. La partie entière du nombre est convertie comme indiqué précédemment ; la partie fractionnaire du nombre est convertie par **multiplications successives** : on multiplie successivement la partie fractionnaire par la base cible, en retenant les parties entières qui apparaissent au fur et à mesure.

Exemple 1 : conversion de $N_{(10)} = 12,925$ en base 2

- partie entière : $12_{(10)} = 1100_{(2)}$
- partie fractionnaire :

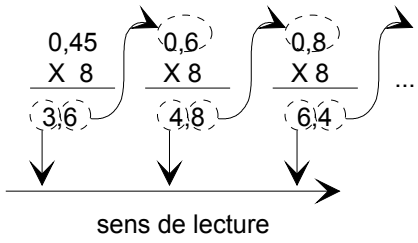


$0,925_{(10)} = 0,111\ldots_{(2)}$

figure 1.3 : conversion de $0,925_{(10)}$ en base 2 par multiplications successives

Finalement $12,925_{(10)} = 1100,111\ldots_{(2)}$

Exemple 2 : conversion de $N_{(10)} = 0,45$ en base 8



$0,45_{(10)} = 0,346\ldots_{(8)}$

figure 1.4 : conversion de $0,45_{(10)}$ en base 8 par multiplications successives

Il est visible sur les deux exemples précédents que la conversion peut ne pas se terminer et que l'on obtient, en s'arrêtant à un nombre fini de positions une approximation de la représentation du nombre.

N. B. Problème du maintien de la résolution lors d'un changement de base.

Soit $(a_n a_{n-1}\ldots a_0, a_{-1}\ldots a_{-m})_{(10)}$ et $(a_p a_{p-1}\ldots a_0, a_{-1}\ldots a_{-k})_{(b)}$ les numérotations de position d'un même nombre *N* exprimé respectivement en base 10 et en base *b*. La résolution est conservée lors du passage de la base 10 à la base b si et seulement si $b^{-k} \leq 10^{-m}$, c'est-à-dire si $k \log b \geq m \log 10$, soit

$$\boxed{k \geq m \frac{\log 10}{\log b}}$$

Exemple 1 : pour conserver la résolution lors du passage de $0,925_{(10)}$ en base 2, il faut garder $k \geq 3 \frac{\log 10}{\log 2} \approx 9,97$, soit 10 bits après la virgule.

Exemple 2 : pour conserver la résolution lors de la conversion de $0,45_{(10)}$ en base 8, il faut garder

$k \geq 2\dfrac{\log 10}{\log 8} \approx 2{,}2$, soit 3 bits après la virgule.

2.2.3 Base 2 vers base 2^n

L'utilisation des bases 2^n (8 et 16) permet de réduire le nombre de symboles à écrire tout en conservant la possibilité de conversion instantanée en binaire.

Pour convertir un nombre de la base 2 vers la base 2^n, il suffit de regrouper les bits par groupes de n (3 pour la base octale et 4 pour la base hexadécimale), et de remplacer chacun de ces groupes par le symbole correspondant dans la base d'arrivée. En effet, considérons par exemple un nombre exprimé en binaire sur 12 bits :

$$N = a_{11}2^{11} + a_{10}2^{10} + a_9 2^9 + a_8 2^8 + a_7 2^7 + a_6 2^6 + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0,$$

en regroupant les bits par 4, on obtient

$$\begin{aligned} N &= (a_{11}2^3 + a_{10}2^2 + a_9 2^1 + a_8 2^0).(2^4)^2 + (a_7 2^3 + a_6 2^2 + a_5 2^1 + a_4 2^0).(2^4)^1 + (a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0).(2^4)^0 \\ &= (a_{11}2^3 + a_{10}2^2 + a_9 2^1 + a_8 2^0).16^2 + (a_7 2^3 + a_6 2^2 + a_5 2^1 + a_4 2^0).16^1 + (a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0).16^0 \end{aligned}$$

la transformation est alors immédiate.

Pour la partie entière, le regroupement part du bit de poids le plus faible, et pour la partie fractionnaire, du bit de poids le plus fort (de la virgule). Lorsqu'un groupe est incomplet, on le complète avec des 0.

Exemple 1 : conversion de $N_{(2)} = 1100111010101$ en base 8 puis 16

Base 8 : $N = 1\ 100\ 111\ 010\ 101_{(2)} = \underbrace{001}_{1_{(8)}}\ \underbrace{100}_{4_{(8)}}\ \underbrace{111}_{7_{(8)}}\ \underbrace{010}_{2_{(8)}}\ \underbrace{101}_{5_{(8)}} = 14725_{(8)}$,

Base 16 : $N = 1\ 1001\ 1101\ 0101_{(2)} = \underbrace{0001}_{1_{(16)}}\ \underbrace{1001}_{9_{(16)}}\ \underbrace{1101}_{\mathrm{D}_{(16)}}\ \underbrace{0101}_{5_{(16)}} = 19\mathrm{D}5_{(16)}$.

Exemple 2 : conversion de $N_{(2)} = 110100110{,}101101$ en base 8 puis 16

Base 8 : $N = 110\ 100\ 110{,}101\ 101_{(2)} = \underbrace{110}_{6_{(8)}}\ \underbrace{100}_{4_{(8)}}\ \underbrace{110}_{6_{(8)}},\underbrace{101}_{5_{(8)}}\ \underbrace{101}_{5_{(8)}} = 646{,}55_{(8)}$

Base 16 : $N = 1\ 1010\ 0110{,}1011\ 01_{(2)} = \underbrace{0001}_{1_{(16)}}\ \underbrace{1010}_{\mathrm{A}_{(16)}}\ \underbrace{0110}_{6_{(16)}},\underbrace{1011}_{\mathrm{B}_{(16)}}\ \underbrace{0100}_{4_{(16)}} = 1\mathrm{A}6,\mathrm{B}4_{(16)}$

2.2.4 Base 2^n vers base 2

Pour la conversion inverse, il suffit de développer chaque symbole de la représentation dans la base 2^n sur n bits.

Exemple 1 : $4A1_{(16)} = \underbrace{0100}_{4 \text{ en base } 2} \ \underbrace{1010}_{A \text{ en base } 2} \ \underbrace{0001}_{1 \text{ en base } 2} = 010010100001_{(2)}$.

Exemple 2 : $273{,}15_{(8)} = 010\ 111\ 011{,}001\ 101 = 10111011{,}001101_{(2)}$

2.3 Représentation binaire des nombres signés

Les systèmes numériques doivent être capables de traiter des nombres positifs et négatifs. L'utilisation d'une représentation signée suppose l'utilisation d'un **format** (nombre de bits) fixé au préalable.

2.3.1 Représentation en complément à 2

Le **complément à 2** est le mode de représentation le plus utilisé en arithmétique binaire et donc dans les ordinateurs pour coder les nombres entiers.

Dans cette représentation, les nombres positifs se représentent par leur valeur binaire naturelle. Par exemple +6 est représenté par 0000 0110 sur un format de 8 bits.

La représentation des nombres négatifs s'obtient comme suit :

- On part de la représentation binaire naturelle de l'opposé arithmétique du nombre à coder (nombre positif),
- On calcule son **complément à 1** (CA1) ou **complément restreint**. Celui-ci est obtenu en inversant tous ses bits,
- On en déduit son **complément à 2** (CA2) ou **complément vrai** en ajoutant 1 au niveau du LSB.

Exemple : représentation de -5 en CA2 sur un format de 8 bits

- Représentation binaire naturelle de +5 : 5 = 0000 0101,
- CA1 de +5 : $\overline{5} = 1111\ 1010$,
- CA2 de +5 : $-5 = 1111\ 1011$.

On identifie le CA2 d'un nombre à son opposé arithmétique car $A + (-A) = 2^n = 0 \bmod 2^n$, si n est le format de représentation du nombre A. En effet, soit $A = a_{n-1} \ldots a_1 a_0$, alors $\overline{A} = \overline{a_{n-1}} \ldots \overline{a_1}\ \overline{a_0}$, et donc $A + \overline{A} = 11 \ldots 11$, soit $A + \overline{A} = 2^n - 1$, et $-A = \overline{A} + 1$.

La représentation en complément à 2 présente les caractéristiques suivantes :

- Le principe d'obtention de l'opposé d'un nombre négatif est le même que celui permettant d'obtenir l'opposé d'un nombre positif,
- Le nombre 0 a une représentation unique,
- Un format sur n bits permet de coder en CA2 les nombres N vérifiant

$$-2^{n-1} \leq N \leq +2^{n-1} - 1$$

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$\overline{N}_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	0000	1111	0000	0
1	0001	1110	1111	-1
2	0010	1101	1110	-2
3	0011	1100	1101	-3
4	0100	1011	1100	-4
5	0101	1010	1011	-5
6	0110	1001	1010	-6
7	0111	1000	1001	-7
			1000	-8

tableau 1 : représentation en complément à 2 sur 4 bits

On peut ainsi représenter des nombres compris entre -4 et +3 sur un format de 4 bits, entre -16 et +15 sur un format de 5 bits, entre -32 et +31 sur un format de 6 bits, entre -64 et +63 sur un format de 7 bits, etc.

- Le bit de poids fort (MSB) est représentatif du bit de signe, mais il est traité comme les autres bits dans les opérations arithmétiques : si MSB = 0 le nombre est positif, si MSB = 1 le nombre est négatif.
- Le nombre $+2^{n-1}$ n'est pas représenté. En effet, dans le cas où $n = 4$, le calcul du CA2 de 1000 donne -(-8) = 0111 + 1 = 1000 = -8. Ce qui est arithmétiquement incorrect, car 0 est le seul entier à être son propre opposé. On a donc choisi de supprimer la représentation du nombre $+2^{n-1}$, un MSB à 1 étant représentatif d'un nombre négatif.

N. B. Extension d'un nombre codé en CA2

L'extension d'un nombre codé sur n bits à un format sur $n+k$ bits est réalisé comme suit :

- Si le nombre est positif, on complète les k bits de poids forts par des 0. Par exemple,

$$\underbrace{0110_{(CA2)}}_{\substack{6_{(10)} \text{ codé} \\ \text{sur 4 bits}}} \xrightarrow{6 \text{ bits}} \underline{00}0110_{(CA2)}.$$

- Si le nombre est négatif, on complète les *k* bits de poids forts avec des 1. Par exemple,

$$\underbrace{1010_{(CA2)}}_{\substack{-6_{(10)}\text{ codé}\\ \text{sur 4 bits}}} \xrightarrow{6\text{ bits}} \underline{1}11010_{(CA2)}.$$

2.3.2 Représentation module + signe

Il s'agit d'une représentation parfois utilisée car plus simple que celle du CA2, mais qui est moins bien adaptée aux opérations arithmétiques.

Dans cette représentation, le bit de poids le plus fort représente le signe (MSB = 0 => nombre positif, MSB = 1 => nombre négatif), et les autres bits la valeur absolue du nombre. Ainsi, un format de *n* bits permet de coder les nombres compris entre $-(2^{n-1}-1)$ et $2^{n-1}-1$. Dans cette représentation, le zéro possède deux notations possibles.

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	0000	1000	0
1	0001	1001	-1
2	0010	1010	-2
3	0011	1011	-3
4	0100	1100	-4
5	0101	1101	-5
6	0110	1110	-6
7	0111	1111	-7

tableau 2 : représentation "module + signe" sur 4 bits

N. B. Extension d'un nombre en représentation "module + signe"

L'extension d'un nombre codé sur *n* bits à un format sur *n*+*k* bits consiste à décaler le bit de signe à la position du MSB et à compléter les autres positions par des 0, que le nombre soit positif ou négatif. Par exemple

$$\underbrace{0110_{(M+S)}}_{\substack{6_{(10)}\text{ codé}\\ \text{sur 4 bits}}} \xrightarrow{6\text{ bits}} 0\underset{\sim}{0}0110_{(M+S)}, \text{ et } \underbrace{1110_{(M+S)}}_{\substack{-6_{(10)}\text{ codé}\\ \text{sur 4 bits}}} \xrightarrow{6\text{ bits}} 1\underset{\sim}{0}0110_{(M+S)}.$$
$$\qquad S \qquad\qquad\qquad\qquad S$$

2.3.3 Représentation binaire décalée

Cette représentation peut être déduite du complément à 2 par une simple inversion du bit de signe (MSB = 0 => nombre négatif, MSB = 1 => nombre positif). Cette représentation est commode pour la conversion numérique/analogique car la valeur maximale positive est codée par tous les bits à 1 et la valeur minimale négative par tous les bits à 0.

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	1000	1000	0
1	1001	0111	-1
2	1010	0110	-2
3	1011	0101	-3
4	1100	0100	-4
5	1101	0011	-5
6	1110	0010	-6
7	1111	0001	-7
		0000	-8

tableau 3 : représentation binaire décalée sur 4 bits

2.4 Représentation des nombres fractionnaires

Dans les calculateurs, deux représentations sont utilisées pour représenter les nombres fractionnaires : le codage en virgule fixe et le codage en virgule flottante.

2.4.1 Codage en virgule fixe

Dans cette représentation, les nombres réels sont représentés par des entiers, après avoir décidé d'un facteur d'échelle k qui est une puissance de la base dans laquelle on écrit les entiers. Autrement dit, un bloc de n bits est considéré comme un nombre dont la partie entière et le signe sont codés sur $n - k$ bits, et dont la partie fractionnaire est codée sur k bits. La résolution d'une telle représentation est de 2^{-k}. L'addition de deux nombres réels en virgule fixe (s'ils possèdent le même facteur d'échelle k) revient à additionner les deux entiers qui représentent ces nombres. Ce codage est surtout utilisé dans les processeurs de traitement de signal (DSP) où les exigences de rapidité sont primordiales. En revanche, la dynamique de cette représentation est assez limitée : pour un format sur n bits avec k bits après la virgule, la dynamique est de $(2^n - 1) / 2^k$.

2.4.2 Codage en virgule flottante

Dans le cas précédent, le facteur d'échelle était fixe. Dans le cas du codage en virgule flottante, le facteur d'échelle peut varier au cours du calcul. On utilise l'écriture semi-logarithmique suivante :

$$N = S.M.b^e \qquad \text{(équation 2)}$$

avec :

- S : signe du nombre,
- M : mantisse,
- b : base de numération (ici $b = 2$),
- e : exposant.

L'exposant e est représentatif de l'ordre de grandeur du nombre et la mantisse M est représentative de sa précision. Le terme b^e joue le rôle de facteur d'échelle de la représentation, mais il est ici *explicite*, contrairement au cas précédent.

Exemple : représentation normalisée IEEE simple précision des nombres flottants en machine sur 32 bits (Standard IEEE 754-1985)

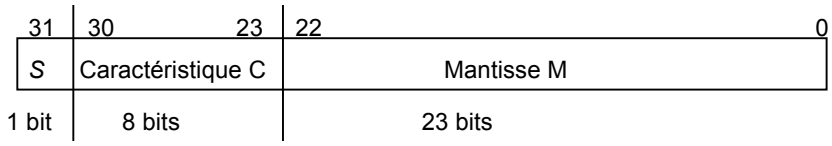


figure 1.5 : norme IEEE, simple précision sur 32 bits

Chaque valeur à représenter est dans ce cas déterminée par l'expression suivante :

$$N = (-1)^S \times 2^{C-127} \times 1,M \qquad \text{(équation 3)}$$

où la valeur de l'exposant e est donnée par $e = C - 127$. L'addition de 127 à e permet de coder l'exposant en binaire naturel et de s'affranchir du problème de son signe.

Cette représentation présente des configurations particulières :

- $C = 0$ et $M = 0$: nombre zéro,
- $C = 255$ et $M = 0$: nombre infini ($+\infty$ si $S = 0$, $-\infty$ si $S = 1$)
- $C = 0$ et $M \neq 0$: nombre dénormalisé (très petit),
- $C = 255$ et $M \neq 0$: code réservé (non interprété comme un nombre).

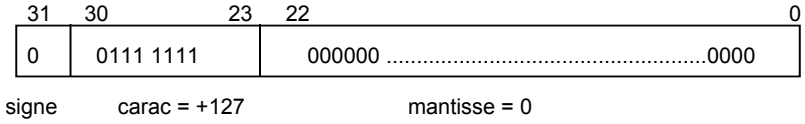
Exercice : traduire le nombre décimal $1_{(10)}$ dans le format précédemment présenté.

Il faut le mettre sous la forme normalisée de l'équation 3, soit

$$1 = (-1)^0 \times 2^{127-127} \times 1{,}00 \cdots 0$$

d'où $S = 0$, $C = 127$, et $M = 000.....0$.

On en déduit la représentation en virgule flottante :



soit $3F800000_{(16)}$.

La représentation en virgule flottante autorise une dynamique plus grande qu'en virgule fixe. Dans le format précédent, on peut coder des nombres dont la valeur absolue va de 10^{-38} à 10^{38}, et le standard d'écriture en virgule flottante double précision (sur 64 bits) permet de représenter des nombres atteignant 10^{300}.

2.5 Classification des codes binaires

Le champ d'application des systèmes numériques est très étendu. Lorsque l'application ne nécessite pas de calculs arithmétiques, les codages précédents sont inutiles ou peu adaptés. On utilise alors des codages possédant d'autres propriétés. On emploie ainsi dans certains systèmes des codes permettant d'éviter des états transitoires parasites lors de la saisie de données, ou de visualiser facilement des chiffres ou des lettres, ou bien encore de détecter des erreurs et/ou de les corriger dans un résultat susceptible d'être erroné. Nous présentons ci-après quelques codes fréquemment utilisés.

L'ensemble des codes binaires peuvent être regroupés en deux classes : les **codes pondérés** et les **codes non pondérés**.

2.5.1 Codes pondérés

Un code est dit pondéré si la position de chaque symbole dans chaque mot correspond à un poids fixé : par exemple 1, 10, 100, 1000 ... pour la numération décimale, et 1, 2, 4, 8, ... pour la numération binaire. Les codes pondérés ont, en général, des propriétés intéressantes du point de vue arithmétique.

2.5.1.1 Le code binaire pur et ses dérivés (octal, hexadécimal)

Ce sont les codes utilisés en arithmétique binaire et qui ont été étudiés dans la première partie de ce chapitre.

2.5.1.2 Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal)

Ce code est utilisé dans de nombreux systèmes d'affichage, de comptage ou même les calculatrices de poche. Dans le code BCD chaque chiffre d'un nombre décimal (de $0_{(10)}$ à $9_{(10)}$) est codé à l'aide de 4 bits (de $0000_{(2)}$ à $1001_{(2)}$). Ainsi le code BCD n'utilise que 10 **mots de codes** de 4 bits. Par exemple la représentation du nombre $1995_{(10)}$ est : $(0001\ 1001\ 1001\ 0101)_{(BCD)}$. Il est possible d'effectuer des opérations arithmétiques en BCD, mais celles-ci sont plus complexes qu'en binaire classique. Ce code est pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100, ...

2.5.2 Codes non pondérés

Dans le cas des codes non pondérés, il n'y a pas de poids affecté à chaque position des symboles. On convient simplement d'un tableau de correspondance entre les objets à coder et une représentation binaire. De tels codes peuvent néanmoins parfois posséder des propriétés arithmétiques intéressantes, comme le code **excédent 3**.

2.5.2.1 Code excédent 3 ou excess 3

Le code excédent 3 utilise, tout comme le code BCD, 10 mots de codes, auxquels on fait correspondre les 10 chiffres décimaux.

$N_{(10)}$	$N_{(XS3)}$			
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

tableau 4 : code excédent 3

Il est obtenu en décalant le code binaire de trois lignes vers le haut. Ce code peut être intéressant pour effectuer des soustractions car le complément à 1 de la représentation binaire d'un chiffre correspond au complément à 9 de ce chiffre. Ainsi, toute opération de soustraction se ramène à une addition.

Par exemple $5_{(XS3)} = 1000$, son complément à 1 est obtenu par inversion des bits, $\overline{5}_{(XS3)} = 0111 = 4_{(XS3)}$, le résultat en excédent 3 est le nombre 4, qui est le complément à 9 de 5 en décimal. Ainsi, pour faire une soustraction, il suffit d'ajouter le complément à 1 du nombre à retrancher, puis 1. Par exemple, $(7-5)_{(XS3)} = 7_{(XS3)} + \overline{5}_{(XS3)} + 1_{(XS3)} = 1010 + 0111 + 0100 = 0101 = 2_{(XS3)}$.

Le code excédent 3 ne présente pas d'intérêt en addition.

2.5.2.2 Code binaire réfléchi ou code de Gray

Ce code numérique n'étant pas pondéré, il est peu employé pour les opérations arithmétiques. Il est, par contre, utilisé pour le codage des déplacements angulaires, linéaires ou pour la réalisation des tableaux de Karnaugh (cf. chapitre « Propriétés des variables et fonctions logiques »). La propriété principale de ce code est que **deux mots successifs du code ne diffèrent que par un élément binaire**. Ceci permet, d'une part d'éviter la génération d'aléas (états parasites) au passage de deux combinaisons successives, et d'autre part de tirer parti de cette adjacence du codage pour simplifier les fonctions.

L'appellation "binaire réfléchi" provient de sa technique de construction : on peut construire un code de Gray sur *n* bits à partir d'un code de Gray sur *n*-1 bits en procédant comme suit : on copie les mots du code de départ, précédés d'un 0, suivis des mots du même code, pris dans **l'ordre inverse** et précédés d'un 1. Ceci permet de construire un code de Gray de n'importe quel format (cf. figure 1.6).

sur 1 bit
0
1

sur 2 bits	
0	0
0	1
1	1
1	0

sur 3 bits		
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

sur 4 bits			
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

figure 1.6 : construction du code de Gray sur 1, 2, 3, et 4 bits

2.5.2.3 Codes redondants

Il existe un ensemble de codes conçus pour pouvoir détecter, voire corriger des erreurs dans des messages binaires. Leur principe repose sur l'insertion de données redondantes dans l'information initiale. Leur étude approfondie relève du domaine des communications numériques, et ne sera pas traité dans ce cours. Nous citerons cependant quelques exemples simples de codes redondants, les codes *p* parmi *n* et les codes de contrôle de parité.

- **Code *p* parmi *n***

Ce code est constitué de $\mathbf{C}_n^p = \frac{n!}{p!(n-p)!}$ mots de code. Chaque mot de code est codé sur *n* bits et contient exactement *p* "1" et (*n* - *p*) "0". Par exemple, le code 2 parmi 5 (tableau 5) est constitué de 10 mots de codes et permet de coder les chiffres décimaux.

$N_{(10)}$	$N_{(2 \text{ parmi } 5)}$				
0	0	0	0	1	1
1	1	1	0	0	0
2	1	0	1	0	0
3	0	1	1	0	0
4	1	0	0	1	0
5	0	1	0	1	0
6	0	0	1	1	0
7	1	0	0	0	1
8	0	1	0	0	1
9	0	0	1	0	1

tableau 5 : code 2 parmi 5

L'utilisation de ce code permet, à la réception d'une information, de vérifier par comptage du nombre de 1 si une erreur s'est introduite dans l'information transmise. Dans le cas où plus d'une erreur s'est glissée dans un mot de code, la détection n'est pas assurée dans tous les cas. Ce code ne permet pas non plus de trouver la place de l'erreur, donc de la corriger. D'autre part, le décodage des combinaisons est particulièrement simple, car il ne porte que sur 2 bits par combinaison.

- **Code contrôle de parité**

Le codage d'un mot de *n* bits par contrôle de parité consiste à y adjoindre un $(n+1)^{\text{ème}}$ bit dont le rôle est de rendre systématiquement pair ou impair le nombre total de 1 contenus dans l'information codée. Si une erreur se glisse dans l'information, le nombre de 1 devient impair et l'erreur est détectée. Ce code ne permet pas non plus de corriger les erreurs.

2.5.3 Codes alphanumériques

Certains codes peuvent avoir une signification non numérique. Le plus connu d'entre eux est le code ASCII (American Standard Code for Information Interchange), qui est utilisé pour représenter les caractères alphanumériques. Dans ce code, 128 combinaisons (lettres, chiffres, signes de ponctuation, caractères de contrôle, etc.) sont codées à l'aide de 7 bits. Les transmissions asynchrones entre machines s'effectuant souvent sur un format de 8 bits, le dernier bit est alors utilisé pour contrôler la parité du message.

3. Opérations arithmétiques

3.1 Addition et soustraction

Le principe de l'addition est dans toutes les bases similaire à celui de l'addition décimale : on additionne symbole par symbole en partant des poids faibles, et en propageant éventuellement une retenue.

Si le format des nombres est fixe, le résultat de l'addition peut donner lieu à un dépassement de capacité. Par exemple, le résultat de l'addition de 2 nombres binaires codés sur n bits peut dépasser la plus grande valeur codable sur n bits ($2^n - 1$ en binaire naturel).

La soustraction, en arithmétique binaire, est le plus souvent appliquée sur des nombres signés. Dans ce cas, cette opération se ramène dans tous les cas à une addition.

Dans le cas où les nombres sont codés en complément à 2, l'addition de 2 nombres exprimés sur n bits fournit toujours un résultat correct, sauf dans le cas où le résultat n'est pas représentable sur les n bits. Il y a alors dépassement de capacité lorsque les deux opérandes sont de même signe et que le résultat est de signe opposé. Dans le registre d'état d'un ordinateur, deux indicateurs viennent renseigner le programmeur (ou le système d'exploitation) sur la validité des résultats obtenus : la retenue (carry C) et le débordement (overflow OVF). L'indicateur C signale la présence d'une retenue au niveau des poids forts; l'indicateur OVF indique que le résultat de l'opération n'est pas représentable dans le système du complément à 2 sur le format défini au départ. Nous allons illustrer le positionnement de la retenue et du débordement par quatre exemples :

0000 0110 (+6)	0111 1111 (+127)	0000 0100 (+4)	0000 0100 (+4)
+ 0000 0100 (+4)	+ 0000 0001 (+1)	+ 1111 1110 (-2)	+ 1111 1100 (-4)
0000 1010 (+10)	1000 0000 (-128)	1 0000 0010 (+2)	1 0000 0000 (0)
OVF=0	OVF=1	OVF=0	OVF=0
C=0	C=0	C=1 ignoré	C=1 ignoré
résultat correct	résultat incorrect	résultat correct	résultat correct

Pour que le résultat d'une opération sur n bits soit correct dans la méthode du complément à 2, il faut que les retenues de rang n et de rang n+1 soient identiques.

3.2 Multiplication et division

Les opérations de multiplication et de division sont plus complexes que l'addition, et ne sont traitées que partiellement dans ce chapitre.

3.2.1 Multiplication et division par 2^k

Le nombre 2 occupant une place privilégié en numération binaire (tout comme le nombre 10 en numération décimale), les cas de la multiplication et de la division par 2^k peuvent être traités à part.

- **Multiplication par** 2^k

Multiplier un nombre binaire par 2^k consiste à décaler la virgule de k positions vers la droite, ou à ajouter k "0" au niveau des bits de poids faible dans le cas des entiers. Par exemple, soit $N = 18_{(10)} = 10010_{(2)}$. La multiplication de N par 2 donne $N.2_{(10)} = 36_{(10)} = 100100_{(2)}$, et sa multiplication par 4, $N.4_{(10)} = 72_{(10)} = 1001000_{(2)}$. Le mécanisme est le même que celui appliqué lors de la multiplication d'un nombre décimal par 10^k.

- **Division par** 2^k

Le mécanisme est inverse de celui de la multiplication. Diviser un nombre binaire par 2^k consiste à décaler la virgule de k positions vers la gauche, ou à enlever les k bits de poids faible dans le cas d'une division entière. Par exemple, soit $N = 37_{(10)} = 100101_{(2)}$. La division entière de N par 4 donne $N/4_{(10)} = 9_{(10)} = 1001_{(2)}$, tandis que la même division considérée sur des réels donne $N/4_{(10)} = 9{,}25_{(10)} = 1001{,}01_{(2)}$.

3.2.2 Cas général

La multiplication de deux nombres binaires de n bits fournit un résultat sur $2n$ bits. Lorsque les nombres ne sont pas signés, le principe est le même qu'en décimal et fait intervenir des produits partiels de bits, des additions et des décalages. A titre d'exemple, la multiplication de 2 nombres de 4 bits non signés, A et B, se décompose comme suit :

				A_3	A_2	A_1	A_0	Multiplicande A
			$\times$	B_3	B_2	B_1	B_0	Multiplieur B
				A_3B_0	A_2B_0	A_1B_0	A_0B_0	Produit partiel $A \times B_0$
			A_3B_1	A_2B_1	A_1B_1	A_0B_1		$A \times B_1$ décalé de 1 rang
		A_3B_2	A_2B_2	A_1B_2	A_0B_2			$A \times B_2$ décalé de 2 rangs
	A_3B_3	A_2B_3	A_1B_3	A_0B_3				$A \times B_3$ décalé de 3 rangs
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	Produit $P = A \times B$

La multiplication de A par B se déroule en trois étapes :

- Multiplication de A par chacun des symboles de B : en base 2, la multiplication de A par le symbole B_i revient à faire un ET logique entre chaque symbole de A et B_i,
- Décalage de $A \times B_i$ de i rangs vers la gauche,
- Addition des résultats de l'étape précédente.

Lorsque les nombres à multiplier sont signés, le principe de l'opération devient plus complexe et fait appel à des algorithmes non traités dans ce cours (cf. par exemple [Aum96]). Néanmoins, dans le cas d'une représentation en complément à 2, l'algorithme de multiplication précédent est applicable, moyennant une légère modification, si le multiplieur est positif :

- Si le multiplicande est positif, pas de changement,
- Si le multiplicande est négatif, tous les produits partiels sont négatifs ou nuls. Il faut étendre les produits partiels non nuls à 2*n* bits en rajoutant des 1 sur les bits de poids forts.

Exemple :

				1	0	1	1	(-5)
			×	0	0	1	1	(+3)
1	1	1	1	1	0	1	1	
1	1	1	1	0	1	1		
0	0	0	0	0	0			
0	0	0	0	0				
1	1	1	1	0	0	0	1	(-15)

Dans les systèmes numériques, la division binaire n'est pas réalisée suivant la méthode utilisée en décimal [Aum96].

4. Conclusion

Ce chapitre présente les systèmes de numération et de codage les plus utilisés dans les systèmes numériques. La numération fait encore, néanmoins, l'objet de recherches. De nouveaux codages des nombres, réels notamment, sont à l'étude pour effectuer plus rapidement certaines opérations et améliorer les performances des opérateurs [Mul89]. La réalisation des opérations de codage, décodage, conversion de codes ainsi que des opérations arithmétiques est étudiée dans le chapitre 4, intitulé « Fonctions et circuits combinatoires ».

5. Bibliographie

[Aum96] M. Aumiaux, *Logique arithmétique et techniques synchrones*, 1ère partie : *Arithmétique binaire et décimale*, Enseignement de l'électronique, Masson, 1996.

[GR87a] M. Gindre et D. Roux, *Electronique numérique : logique combinatoire et technologie, cours et exercices*, McGraw-Hill, Paris, 1987.

[LV86] J.-C. Lafont et J.-P. Vabre, *Cours et problèmes d'électronique numérique*, Ellipses, 1986.

[Mul89] J.-M. Muller, *Arithmétique des ordinateurs, opérateurs et fonctions élémentaires*, Etudes et recherches en informatique, Masson, 1989.

[TP94] J.-L. Danger, C. Degois, A. Galisson, J. Leroux, D. Roux, *Composants et fonctions de l'électronique numérique, cours*, polycopié Télécom Paris, 1994.

Chapitre 2 : Propriétés des variables et fonctions logiques

1. Introduction

Le fonctionnement des systèmes numériques repose sur la manipulation de variables et fonctions dont les valeurs sont représentées par des grandeurs physiques dites **binaires** car ne pouvant prendre que deux valeurs (généralement notées **0** et **1**). La structure mathématique permettant de formaliser les opérations de manipulation de ces grandeurs binaires est dite **algèbre de commutation** ou plus communément **algèbre de Boole**. Nous nous intéressons dans ce chapitre aux bases et aux propriétés fondamentales de l'algèbre de Boole indispensables à la compréhension du fonctionnement des systèmes numériques.

2. Propriétés de l'algèbre de Boole

2.1 Définitions

Dans l'algèbre de commutation, une variable ne peut prendre que 0 ou 1 comme valeur possible. Une telle variable est dite **variable logique**, **variable binaire**, ou **variable booléenne**. De même, une fonction de n variables logiques ne peut prendre comme valeur que 0 ou 1. Elle est dite **fonction logique**, **fonction binaire**, ou **fonction booléenne**.

2.2 Table de vérité d'une fonction logique

C'est une table donnant l'état logique de la fonction pour chacune des combinaisons des états de ses variables. Une fonction de n variables est représentée par une table de vérité à n+1 colonnes et au plus 2^n lignes. Le tableau 2.1 donne la forme générale d'une fonction de deux variables logiques.

A	B	F(A,B)
0	0	F(0,0)
0	1	F(0,1)
1	0	F(1,0)
1	1	F(1,1)

tableau 2.1 : forme générale de la table de vérité d'une fonction de deux variables logiques

2.3 Les fonctions logiques élémentaires

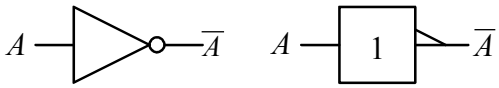
Trois fonctions suffisent pour définir une algèbre de Boole : la **complémentation**, le **produit logique,** et l'**addition logique**.

2.3.1 La fonction de complémentation ou fonction NON

Le complément de la variable A se note $\overline{A}$ (lire « A barre » ou « non A »). $\overline{A}$ vaut 1 (respectivement 0) si et seulement si A vaut 0 (respectivement 1). On parle encore de fonction d'**inversion logique**. Le tableau 2.2 donne la table de vérité de la fonction de complémentation. Les symboles usuellement utilisés pour représenter graphiquement l'opérateur correspondant, appelé **inverseur**, sont ceux de la figure 2.1.

A	$\overline{A}$
0	1
1	0

tableau 2.2 : table de vérité de la fonction NON



(a) (b)

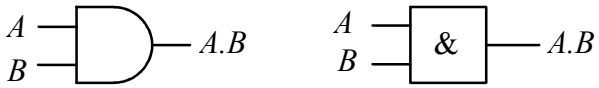
figure 2.1 : symboles logiques d'un inverseur
(a) notation usuelle (ancienne notation US)
(b) notation normalisée IEEE (ancienne notation européenne)

2.3.2 La fonction produit logique ou fonction ET

Le produit logique de 2 variables se note $A.B$, AB, ou bien encore $A \wedge B$ (lire « A et B »). $A.B$ vaut 1 si et seulement si A et B valent 1. Le tableau 2.3 donne la table de vérité de la fonction ET, et la figure 2.2 les symboles logiques de l'opérateur associé.

A	B	$A.B$
0	0	0
0	1	0
1	0	0
1	1	1

tableau 2.3 : table de vérité de la fonction ET



(a) (b)

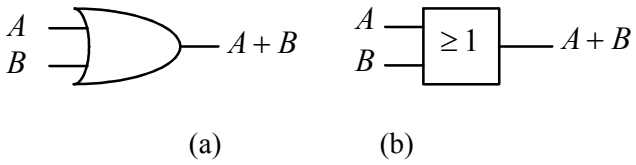
figure 2.2 : symboles logiques de l'opérateur ET.
(a) notation usuelle
(b) notation normalisée IEEE

2.3.3 La fonction addition logique ou fonction OU

L'addition logique de 2 variables se note $A+B$ ou $A \vee B$ (lire « A ou B »). $A+B$ vaut 0 si et seulement si A et B valent 0. Le tableau 2.4 donne la table de vérité de la fonction OU, et la figure 2.3 les symboles logiques de l'opérateur associé.

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

tableau 2.4 : table de vérité de la fonction OU



(a) (b)

figure 2.3 : symboles logiques de l'opérateur OU.
(a) notation usuelle
(b) notation normalisée IEEE

2.3.4 Propriétés des fonctions NON, ET, et OU

- *Commutativité des fonctions ET et OU :*

$$AB = BA$$
$$A + B = B + A$$

- *Associativité des fonctions ET et OU :*

$$A(BC) = (AB)C = ABC$$
$$A + (B + C) = (A + B) + C = A + B + C$$

- *Eléments neutres pour les fonctions ET et OU*

$$A.1 = 1.A = A$$
$$A + 0 = 0 + A = A$$

- *Eléments absorbants pour les fonctions ET et OU*

$$A.0 = 0.A = 0$$
$$A + 1 = 1 + A = 1$$

- *Propriété d'idempotence des fonctions ET et OU*

$$A.A = A$$
$$A + A = A$$

- *Propriétés de l'inversion logique*

$$\overline{\overline{A}} = A$$
$$\overline{A}.A = 0$$
$$\overline{A} + A = 1$$

- *Distributivité de ET par rapport à OU*

$$A(B + C) = AB + AC$$
$$(A + B)C = AC + BC$$

- *Distributivité de OU par rapport à ET*

$$A + BC = (A + B)(A + C)$$
$$AB + C = (A + C)(B + C)$$

- *Autres relations utiles se déduisant des précédentes (relations de simplification)*

$$A + AB = A$$
$$A(A + B) = A$$
$$A + \overline{A}B = A + B$$
$$A(\overline{A} + B) = AB$$

- *Théorème de De Morgan*

$$\overline{A + B} = \overline{A}.\overline{B}$$
$$\overline{A.B} = \overline{A} + \overline{B}$$

Ce théorème se généralise à un nombre quelconque de variables :

$$\overline{\sum_i X_i} = \prod_i \overline{X_i}$$
$$\overline{\prod_i X_i} = \sum_i \overline{X_i}$$

N. B. On notera que l'analogie entre l'addition logique (resp. produit logique) et l'addition (resp. multiplication) de l'arithmétique classique se limite à un nombre très restreint de propriétés.

2.3.5 Opérateurs secondaires

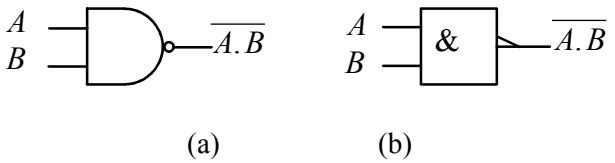
Dans les circuits logiques, on utilise également des opérateurs qui sont des combinaisons des fonctions ET, OU, et NON.

2.3.5.1 La fonction NON ET ou NAND : $\overline{A.B}$

La table de vérité de la fonction NON ET se déduit immédiatement de celle de la fonction ET par inversion du résultat (tableau 2.5).

A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

tableau 2.5 : table de vérité de la fonction NON ET



(a) (b)

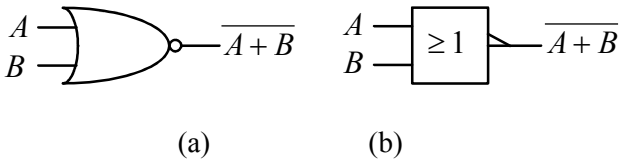
figure 2.4 : symboles logiques de l'opérateur NON ET.
(a) notation usuelle
(b) notation normalisée IEEE

2.3.5.2 La fonction NON OU ou NOR : $\overline{A+B}$

La table de vérité de la fonction NON OU se déduit immédiatement de celle de la fonction OU par inversion du résultat (tableau 2.6).

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

tableau 2.6 : table de vérité de la fonction NON OU



(a) (b)

figure 2.5 : symboles logiques de l'opérateur NON OU
(a) notation usuelle
(b) notation normalisée IEEE

2.3.5.3 Quelques propriétés des fonctions NON ET et NON OU

Les propriétés des fonctions NON ET et NON OU se déduisent des propriétés des fonctions élémentaires NON, ET, et OU.

$$\overline{AB} = \overline{BA} \qquad \overline{A+B} = \overline{B+A}$$

$$\overline{(AB)C} = \overline{A(BC)} = \overline{ABC} \text{ mais } \overline{\overline{AB}C} \neq \overline{A\overline{BC}}$$

$$\overline{(A+B)+C} = \overline{A+(B+C)} = \overline{A+B+C} \text{ mais } \overline{\overline{A+B}+C} \neq \overline{A+\overline{B+C}}$$

$$\overline{A.1} = \overline{A} \qquad \overline{A+1} = 0$$

$$\overline{A.0} = 1 \qquad \overline{A+0} = \overline{A}$$

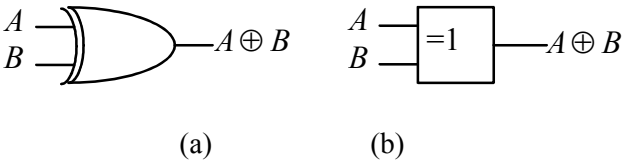
$$\overline{A.A} = \overline{A} \qquad \overline{A+A} = \overline{A}$$

$$\overline{\overline{A}.A} = 1 \qquad \overline{\overline{A}+A} = 0$$

2.3.5.4 La fonction OU exclusif (abrégé OUEX ou XOR) : $A \oplus B = A.\overline{B} + \overline{A}.B$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

tableau 2.7 : table de vérité de la fonction OU exclusif



(a) (b)

figure 2.6 : symboles logiques de l'opérateur OU exclusif.
(a) notation usuelle
(b) notation normalisée IEEE

- *Propriétés de la fonction OU exclusif*

$A \oplus B = B \oplus A$ (commutativité)

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

$A \oplus 0 = A \qquad A \oplus 1 = \overline{A}$

$A \oplus A = 0 \qquad A \oplus \overline{A} = 1$

$A \oplus B = \overline{A} \oplus \overline{B}$

- *Utilisations courantes de la fonction OU exclusif*

⇒ Détection de deux éléments binaires différents,

$A \oplus B = 1 \Leftrightarrow A \neq B$

⇒ Détection d'un nombre de variables impair,

$A \oplus B \oplus C \oplus \ldots = 1 \Leftrightarrow (A, B, C, \ldots)$ contient un nombre impair de 1

⇒ Somme modulo 2 de deux éléments binaires.

2.3.5.5 La fonction ET inclusif (abrégé XNOR) : $A \odot B = \overline{A \oplus B} = A.B + \overline{A}.\overline{B}$

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

tableau 2.8 : table de vérité de la fonction ET inclusif

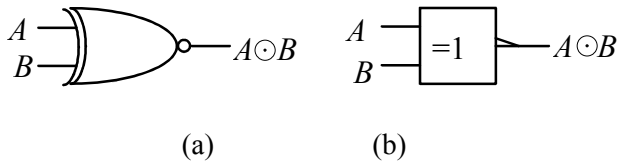


figure 2.7 : symboles logiques de l'opérateur ET inclusif.
(a) notation usuelle
(b) notation normalisée IEEE

- *Propriétés de la fonction ET inclusif*

Les propriétés du ET inclusif se déduisent aisément des propriétés de la fonction OU exclusif en remarquant que

$$A \odot B = \overline{A \oplus B} = A \oplus \overline{B} = \overline{A} \oplus B$$

- *Utilisations courantes de l'opérateur ET inclusif*

⇒ Détection de deux éléments binaires égaux,

$\overline{A \oplus B} = 1 \Leftrightarrow A = B$

⇒ Détection d'un nombre de variables pair,

$\overline{A \oplus B \oplus C \oplus \ldots} = 1 \Leftrightarrow (A, B, C, \ldots)$ contient un nombre pair de 1

2.3.6 Opérateurs complets

Un opérateur logique est dit complet s'il permet de réaliser les trois fonctions de base de l'algèbre de Boole et, par conséquent, toutes les fonctions logiques. Par exemple, l'opérateur NON ET est complet. Il en est de même pour l'opérateur NON OU.

En effet,

$$\overline{A} = \overline{A.A}$$
$$A.B = \overline{\overline{AB}.\overline{AB}}$$
$$A + B = \overline{\overline{A + B}} = \overline{\overline{A}.\overline{B}} = \overline{\overline{AA}.\overline{BB}}$$

De même,

$$\overline{A} = \overline{A + A}$$
$$A.B = \overline{\overline{A.B}} = \overline{\overline{A} + \overline{B}} = \overline{\overline{A + A} + \overline{B + B}}$$
$$A + B = \overline{\overline{A + B} + \overline{A + B}}$$

En revanche, les opérateurs OU exclusif et ET inclusif, ne sont pas complets.

3. Représentation des fonctions logiques

3.1 Formes algébriques disjonctives, conjonctives, canoniques

Considérons la table de vérité de la fonction booléenne F de 3 variables *A*, *B*, et *C*, définie par le tableau 2.9.

A	*B*	*C*	numéro de la combinaison	F(*A*,*B*,*C*)	$\overline{\text{F}(A,B,C)}$
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	2	0	1
0	1	1	3	0	1
1	0	0	4	1	0
1	0	1	5	1	0
1	1	0	6	1	0
1	1	1	7	0	1

tableau 2.9 : table de vérité d'une fonction booléenne F de 3 variables

On peut extraire une expression de F en exprimant les combinaisons des variables A, B, et C pour lesquelles F est égale à 1 : F vaut 1 pour les combinaisons 0, 1, 4, 5, et 6, c'est-à-dire si $\overline{A}\,\overline{B}\,\overline{C}=1$, $\overline{A}\,\overline{B}\,C=1$, $A\overline{B}\,\overline{C}=1$, $A\overline{B}\,C=1$, ou $AB\,\overline{C}=1$. La fonction F peut donc s'écrire sous la forme :

$$\mathrm{F}(A,B,C)=\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}\,C+A\overline{B}\,\overline{C}+A\overline{B}\,C+AB\,\overline{C}$$

L'expression obtenue est une somme logique de produits logiques, il s'agit d'une **forme algébrique disjonctive**, encore appelée **forme ΣΠ**. Les produits logiques font intervenir toutes les variables, sous leur forme directe ou complémentée. Ces produits élémentaires sont appelés **mintermes**. Pour n variables logiques, il existe 2^n mintermes différents, chaque minterme étant égal à 1 pour une seule combinaison des n variables. La représentation d'une fonction sous la forme d'une somme de mintermes est dite **forme canonique disjonctive** ou **première forme canonique**.

On peut extraire une seconde expression de F en exprimant les combinaisons des variables A, B, et C pour lesquelles F est égale à 0. F vaut 0 pour les combinaisons 2, 3, et 7, ce qui peut encore s'écrire :

$$\mathrm{F}(A,B,C)=(A+\overline{B}+C).(A+\overline{B}+\overline{C}).(\overline{A}+\overline{B}+\overline{C})$$

Cette nouvelle expression a une forme duale de la précédente. C'est un produit logique de sommes logiques, il s'agit d'une **forme algébrique conjonctive** ou **forme ΠΣ**. Les sommes logiques composant le produit font intervenir toutes les variables, sous leur forme directe ou complémentée. Elles sont appelées **maxtermes**. Pour n variables logiques, il existe 2^n maxtermes différents, chaque maxterme étant égal à 0 pour une seule combinaison des n variables. La représentation d'une fonction sous la forme d'un produit de maxtermes est dite **forme canonique conjonctive** ou **seconde forme canonique**.

3.2 Représentations de référence d'une fonction logique

Parmi les différentes représentations des fonctions logiques étudiées dans ce chapitre, trois d'entre elles peuvent être considérées comme des représentations de référence en raison de leur unicité :

- la table de vérité,
- les deux formes canoniques.

En effet, deux fonctions logiques sont égales si et seulement si leurs tables de vérité ou leurs formes canoniques sont identiques.

3.3 Critères de choix d'une représentation

L'un des deux types de représentation, forme disjonctive ou conjonctive, peut être préférable à l'autre si des contraintes sont imposées sur la réalisation matérielle des fonctions. En particulier, dans le cas de l'utilisation de **circuits logiques** réalisant les fonctions logiques élémentaires, le type de circuits disponibles peut favoriser une des deux formes.

Ainsi, la forme disjonctive est bien adaptée à une réalisation à base d'opérateurs NON ET. En effet, soit F une fonction de 4 variables écrite sous la forme disjonctive suivante (non canonique dans le cas traité, puisque les produits ne sont pas des mintermes) :

$$\mathrm{F}(A,B,C,D)=A.B+\overline{C}D$$

Pour réaliser cette fonction à l'aide d'opérateurs NON ET et d'inverseurs, il faut dans un premier temps transformer la fonction pour l'écrire sous la forme d'une combinaison de fonctions élementaires NON ET et d'inversion (application du théorème de De Morgan):

$$\mathrm{F}(A,B,C,D) = A.B + \overline{C}D = \overline{\overline{A.B + \overline{C}D}} = \overline{\overline{A.B}.\overline{\overline{C}D}}$$

Il reste ensuite à assembler le nombre adéquat d'opérateurs élémentaires pour réaliser F. La figure 2.8 montre un schéma de réalisation (ou **logigramme**) de F utilisant 3 opérateurs NON ET et 1 inverseur. Si l'opérateur d'inversion n'est pas disponible, il peut lui-même être réalisé à l'aide d'un opérateur NON ET, cf. §2.3.6.

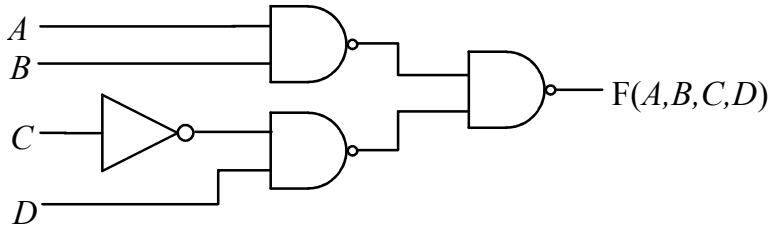


figure 2.8: logigramme de la fonction F à base d'opérateurs NON ET et d'un inverseur

De même, la forme conjonctive est bien adaptée à une réalisation à base d'opérateurs NON OU. En effet, soit une fonction G de 4 variables écrite sous une forme conjonctive :

$$\mathrm{G}(A,B,C,D) = (\overline{A} + \overline{B}).(C + D) = \overline{\overline{(\overline{A} + \overline{B}).(C + D)}} = \overline{\overline{\overline{A} + \overline{B}} + \overline{C + D}}$$

La fonction G peut être réalisée à l'aide de 3 opérateurs NON OU et 2 inverseurs (figure 2.9).

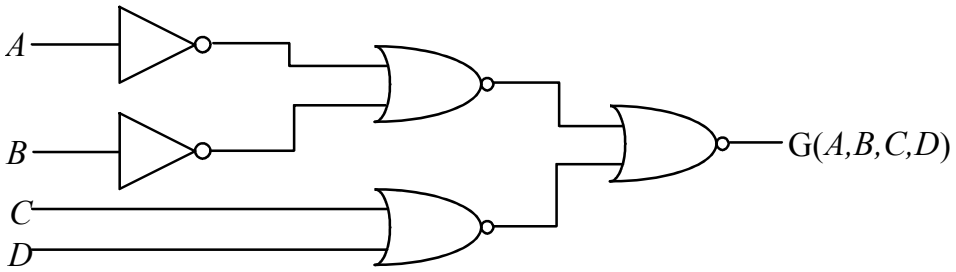


figure 2.9 : logigramme de la fonction G à base d'opérateurs NON OU et d'inverseurs

Lorsqu'aucune contrainte extérieure n'impose l'une des représentations, la forme disjonctive est traditionnellement plus utilisée que la forme conjonctive, en raison de l'analogie de notation entre les opérations logiques et arithmétiques.

Lors de la mise en œuvre d'une fonction logique dans un circuit, deux types de contraintes peuvent être prises en compte : optimiser la **vitesse** du circuit (c.-à-d. obtenir une fréquence maximale de fonctionnement la plus grande possible) ou bien optimiser sa **complexité** (c.-à-d. obtenir un encombrement sur silicium minimal). Dans le cas où la contrainte de **complexité** est la plus forte, il faut utiliser le minimum de matériel. Il est, pour cela, nécessaire de représenter la fonction à réaliser sous une forme simplifiée, c'est-à-dire utilisant un nombre minimal d'opérateurs. Le problème de la simplification des fonctions logiques est traité dans la section suivante.

4. Simplification des fonctions logiques

4.1 Pourquoi simplifier les fonctions logiques ?

Simplifier une fonction logique consiste à rechercher une expression de cette fonction conduisant à la réalisation d'un circuit de coût minimal. Il faut cependant noter que la minimisation à tout prix du nombre d'opérateurs n'est pas toujours le but recherché. Dans le cas de fonctions complexes, des contraintes de vitesse ou de testabilité peuvent aller à l'encontre d'une minimalisation des expressions. On peut, par exemple, être amené à augmenter la complexité des opérateurs d'un circuit pour accroître sa vitesse de fonctionnement, ou à limiter la simplification d'une fonction pour extraire du circuit des variables logiques internes.

4.2 Simplification algébrique

Elle consiste à appliquer les propriétés de l'algèbre de Boole (cf. §2.3.4) aux expressions algébriques des fonctions logiques. Il s'agit principalement de « recettes » dont l'application demande un peu d'entraînement. Nous allons traiter quelques exemples qui permettront de passer en revue la plupart des astuces utilisées pour mener à bien les simplifications.

Dans les exemples suivants, la technique consiste à regrouper judicieusement les termes puis à simplifier en utilisant les relations de simplification vues au §2.3.4.

4.2.1 Exemple 1 : simplification de $F_1 = BC + AC + AB + B$.

$AB + B = B$, donc $F_1 = BC + AC + B$,

d'où $\boxed{F_1 = AC + B}$, car $BC + B = B$.

4.2.2 Exemple 2 : simplification de $F_2 = (A + \overline{B})(A\overline{B} + C)C$.

$(X + C)C = C$, d'où $\boxed{F_2 = (A + \overline{B})C = AC + \overline{B}C}$.

4.2.3 Exemple 3 : simplification de $F_3 = \overline{A}B\overline{C} + AB\overline{C} + ABC + \overline{A}BC$.

Il suffit de remarquer que $AX + \overline{A}X = (A + \overline{A})X = X$, et l'expression devient $\boxed{F_3 = B\overline{C} + BC = B}$.

Dans d'autres cas, il faut « compliquer » l'expression en utilisant les propriétés d'idempotence du ET et du OU, ou les propriétés de l'inversion, pour éliminer des termes superflus.

4.2.4 Exemple 4 : simplification de $F_4 = \overline{A}\,B + AC + BC$.

L'expression reste inchangée si le troisième terme est multiplié par 1 :

$$F_4 = \overline{A}\,B + AC + (A + \overline{A})BC.$$

En utilisant la distributivité de ET par rapport à OU, on obtient

$$\begin{aligned} F_4 &= \overline{A}\,B + AC + ABC + \overline{A}\,BC \\ &= \overline{A}\,B(1+C) + AC(1+B) \end{aligned}$$

d'où $\boxed{F_4 = \overline{A}\,B + AC}$.

4.2.5 Exemple 5 : simplification de $F_5 = (\overline{A} + B)(A + C)(B + C)$.

On ne change pas F_5 en ajoutant 0 à l'un des termes :

$$F_5 = (\overline{A} + B)(A + C)(B + C + \overline{A}\,A).$$

En utilisant ensuite la distributivité de OU par rapport à ET, on obtient

$$\begin{aligned} F_5 &= (\overline{A} + B)(A + C)(\overline{A} + B + C)(A + C + B) \\ &= (\overline{A} + B + 0.C)(A + C + 0.B) \end{aligned}$$

d'où $\boxed{F_5 = (\overline{A} + B)(A + C)}$.

Il peut également être utile de savoir reconnaître le OU exclusif et son complément !

4.2.6 Exemple 6 : simplification de $F_6 = (A\overline{B} + \overline{A}\,B).(AB + \overline{A}\,\overline{B})$.

On remarque que $F_6 = (A \oplus B).(\overline{A \oplus B})$, d'où $\boxed{F_6 = 0}$.

4.2.7 Conclusion

Les méthodes algébriques de simplification présentent un inconvénient majeur : elles ne sont pas systématiques, et leur efficacité dépend donc largement du savoir-faire de la personne qui les applique. Elles ne peuvent, par conséquent, être utilisées que ponctuellement sur des cas simples.

4.3 Simplification par diagramme de Karnaugh

4.3.1 Introduction

Le diagramme ou tableau de Karnaugh est un outil graphique qui permet de simplifier de façon méthodique une fonction logique. Bien que les diagrammes de Karnaugh soient applicables en théorie à des fonctions ayant un nombre quelconque de variables, ils ne sont en pratique utilisables « à la main » que pour un nombre de variables inférieur ou égal à 6.

4.3.2 Adjacence logique

Deux termes sont dits **logiquement adjacents** s'ils ne diffèrent que par une variable. Par exemple, ABC et $\overline{A}BC$ sont deux termes produits adjacents, et $\overline{A}+\overline{B}+\overline{C}+D$ et $\overline{A}+B+\overline{C}+D$ sont deux termes sommes adjacents.

La somme de deux produits adjacents et le produit de deux sommes adjacentes peuvent être simplifiés par mise en facteur, en raison des propriétés de distributivité réciproque des opérateurs ET et OU. En effet,

$$AB + A\overline{B} = A(B + \overline{B}) = A \text{ (distributivité de ET par rapport à OU),}$$

et

$$(A + B)(A + \overline{B}) = A + B\overline{B} = A \text{ (distributivité de OU par rapport à ET).}$$

Un diagramme de Karnaugh est une table de vérité disposée de telle sorte que tous les termes logiquement adjacents soient également géométriquement adjacents, afin de mettre visuellement en évidence les simplifications possibles.

La méthode de Karnaugh est applicable à partir d'une représentation de la fonction sous une de ses deux formes algébriques canoniques. En pratique, la première forme canonique (forme disjonctive) est la plus utilisée. Par la suite, le principe de la simplification est détaillé dans ce cas, mais toutes les étapes décrites sont également applicables pour une représentation sous la forme conjonctive.

4.3.3 Construction d'un diagramme de Karnaugh

Dans un diagramme de Karnaugh, la correspondance entre adjacence logique et adjacence géométrique est due au codage des combinaisons de variables : deux combinaisons voisines ne varient que par un seul bit (codage de Gray, cf. chapitre 1 § 2.5.2.2). Chaque case du tableau représente un minterme, et pour une fonction de n variables, chaque case est adjacente à n autres cases, représentant les n mintermes adjacents.

Lors du remplissage du diagramme, la valeur logique 1 est inscrite dans les cases correspondant aux mintermes présents dans l'expression de la fonction, puis le tableau est complété par des 0. Les 0 peuvent être omis pour alléger l'écriture.

4.3.3.1 Fonction de 2 variables

La figure 2.10(a) donne la position des 4 mintermes dans un tableau de Karnaugh à 2 variables. La figure 2.10(b) montre la correspondance entre adjacence logique et adjacence géométrique : le terme $A\overline{B}$, repéré par le symbole ❶ est adjacent aux termes $\overline{A}\,\overline{B}$ et AB, repérés par le symbole ①. Sur la figure 2.10(c), le tableau est rempli dans le cas de la fonction $F_1 = \overline{A}\,\overline{B} + A\overline{B} + \overline{A}\,B$.

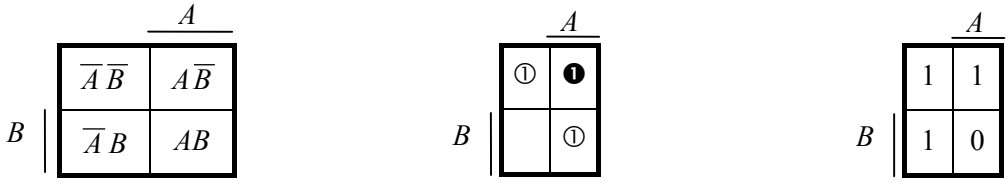


figure 2.10: diagramme de Karnaugh à 2 variables
(a) position des mintermes
(b) termes adjacents à $A\overline{B}$
(c) remplissage pour $F_1 = \overline{A}\,\overline{B} + A\overline{B} + \overline{A}\,B$

4.3.3.2 Fonction de 3 variables

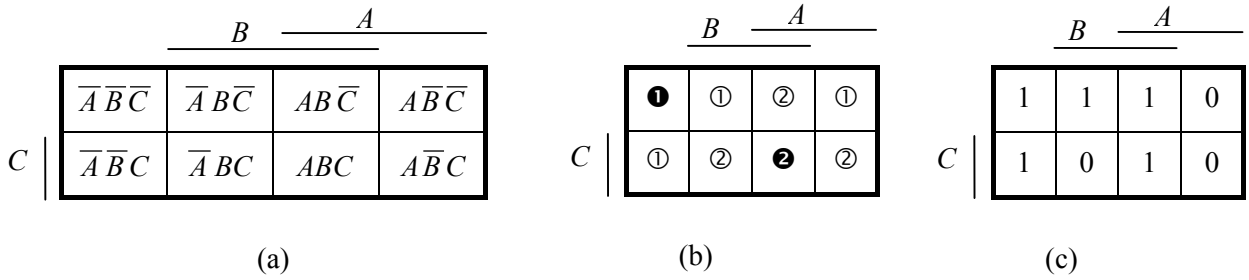


figure 2.11 : diagramme de Karnaugh à 3 variables
(a) position des mintermes
(b) termes adjacents à $\overline{A}\,\overline{B}\,\overline{C}$ (symbole ❶) et à ABC (symbole ❷)
(c) remplissage pour $F_2 = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\overline{C} + AB\,\overline{C} + ABC$

Pour le terme ABC, l'adjacence géométrique est évidente. En revanche, pour retrouver l'adjacence géométrique dans le cas de $\overline{A}\,\overline{B}\,\overline{C}$, il faut remarquer que les cases aux deux extrémités de la première ligne sont adjacentes. Ceci est mis en évidence en représentant le tableau sous forme cylindrique comme le montre la figure 2.12.

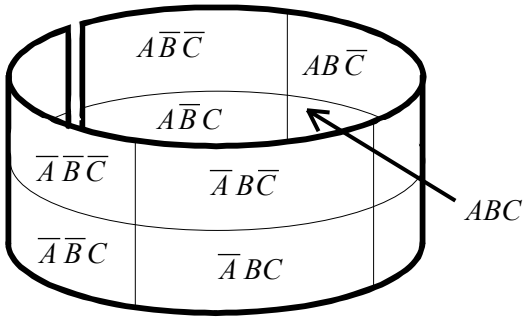


figure 2.12 : représentation cylindrique d' un tableau de Karnaugh à trois variables

Plus généralement, deux cases situées aux extrémités d'une même ligne ou d'une même colonne sont adjacentes. Ceci est dû au code de Gray qui est un code cyclique.

4.3.3.3 Fonction de 4 variables

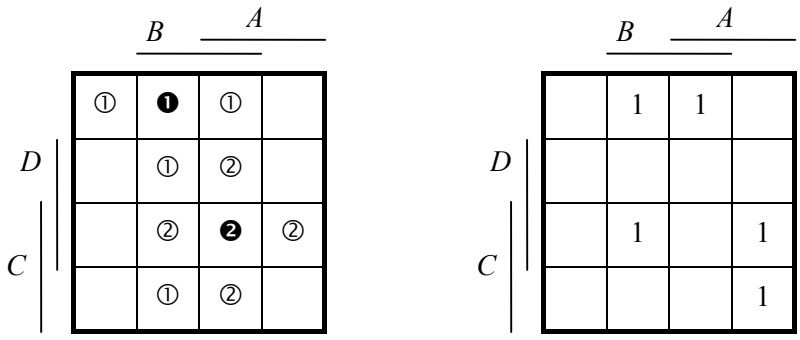


figure 2.13 : diagramme de Karnaugh à 4 variables
(a) termes adjacents à $\overline{A}\,B\overline{C}\overline{D}$ (symbole ❶) et à $ABCD$ (symbole ❷)
(b) remplissage pour $F_3 = \overline{A}\,B\overline{C}\overline{D} + AB\,\overline{C}\overline{D} + \overline{A}\,BCD + A\overline{B}\,CD + A\overline{B}\,C\overline{D}$

Les adjacences sur les bords d'un tableau à quatre variables sont mises en évidence par les deux représentations de la figure 2.14.

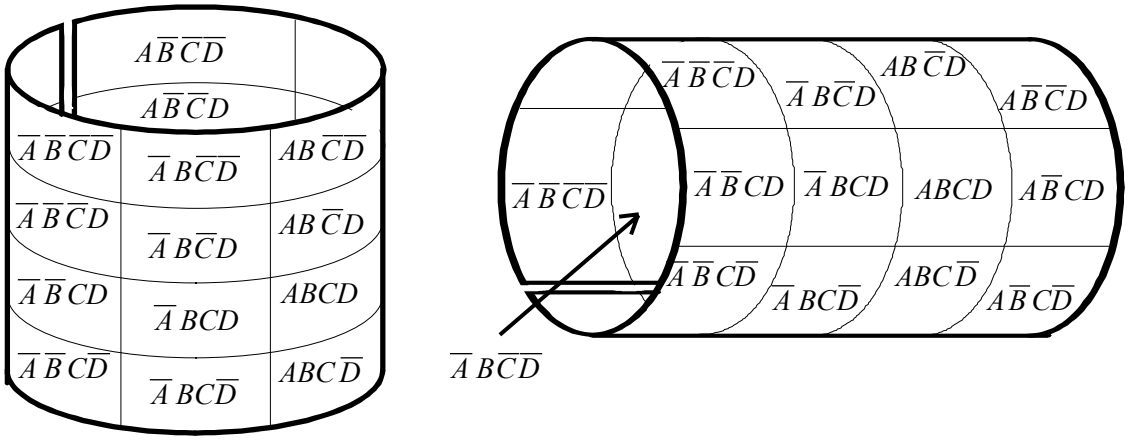


figure 2.14 : représentations cylindriques d'un diagramme de Karnaugh à quatre variables
(a) mise en évidence des adjacences entre lignes
(b) mise en évidence des adjacences entre colonnes

4.3.3.4 Fonctions de 5 et 6 variables

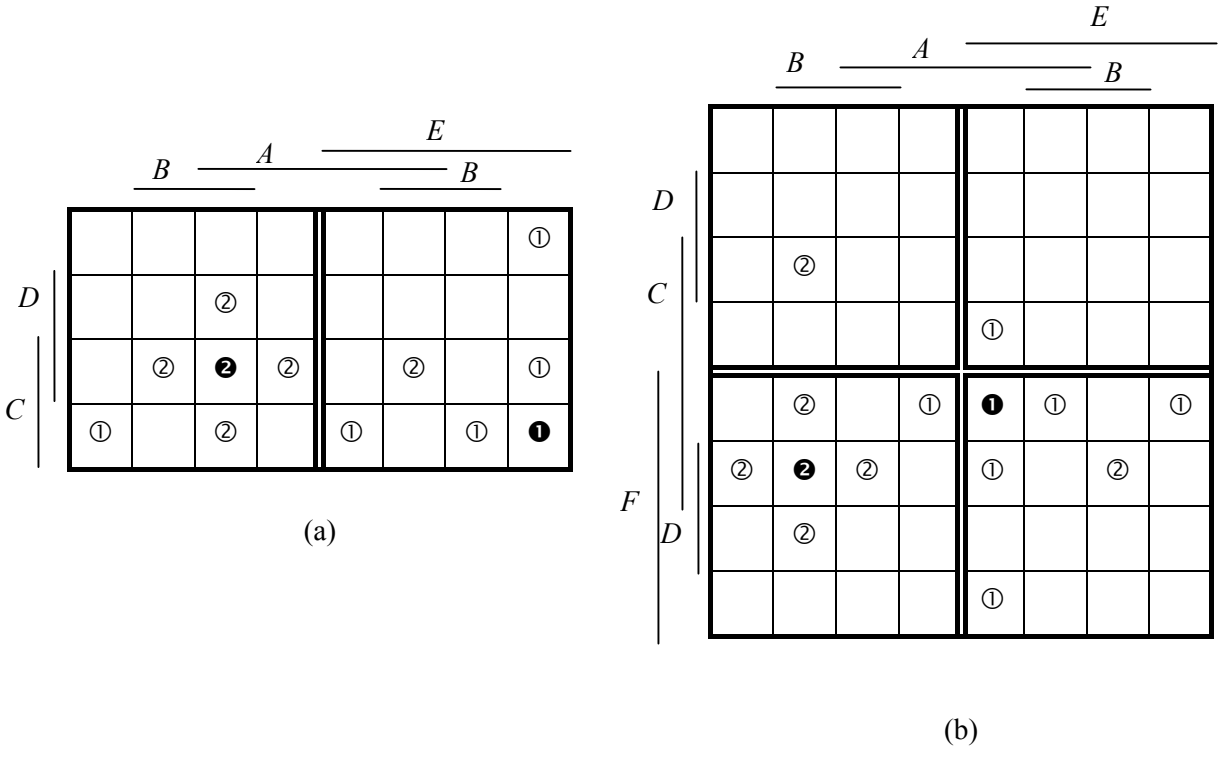


figure 2.15 : forme générale d'un diagramme de Karnaugh à 5 variables (a) et à 6 variables (b)
(a) termes adjacents à $\overline{A}\,\overline{B}\,C\overline{D}E$ (symbole ❶) et à $ABCD\,\overline{E}$ (symbole ❷)
(b) termes adjacents à $A\overline{B}\,C\overline{D}EF$ (symbole ❶) et à $\overline{A}\,BCD\overline{E}F$ (symbole ❷)

On notera qu'à partir de 5 variables, le repérage des termes adjacents devient beaucoup plus délicat, et qu'une représentation similaire à celle de la figure 2.14 est irréalisable. La limite de cette méthode, utilisée « à la main », est donc liée au problème de visualisation des adjacences.

4.3.4 Principe de la simplification

On repère les cases adjacentes contenant un 1 et on les regroupe par paquets de 2^n. Un regroupement par 2^n correspond à la simplification par n variables.

A titre d'exemple, pour une fonction de 3 variables (cf. figure 2.16) :

- 1 case correspond à un minterme donc à un produit des 3 variables,
- 2 cases groupées représentent un produit de 2 variables : il y a simplification par la variable intervenant à la fois sous forme directe et sous forme complémentée. Trois exemples sont présentés dans les tableaux (a), (b), et (c) de la figure 2.16. Ainsi, pour le diagramme (a), $F = \overline{A}\,B\overline{C} + A\overline{B}\,\overline{C} = (\overline{A} + A)B\overline{C} = B\overline{C}$.
- 4 cases groupées représentent un « produit » de 1 variable, comme l'illustrent les diagrammes (d), (e), et (f). Par exemple, le diagramme (d) donne $F = \overline{A}\,B\overline{C} + \overline{A}\,BC + AB\,\overline{C} + ABC = \overline{A}\,B(\overline{C} + C) + AB\,(\overline{C} + C) = \overline{A}\,B + AB = (\overline{A} + A)B = B$. Toutes les variables intervenant à la fois sous forme directe et sous forme complémentée sont éliminées.
- 8 cases regroupées conduisent alors naturellement à F = 1.

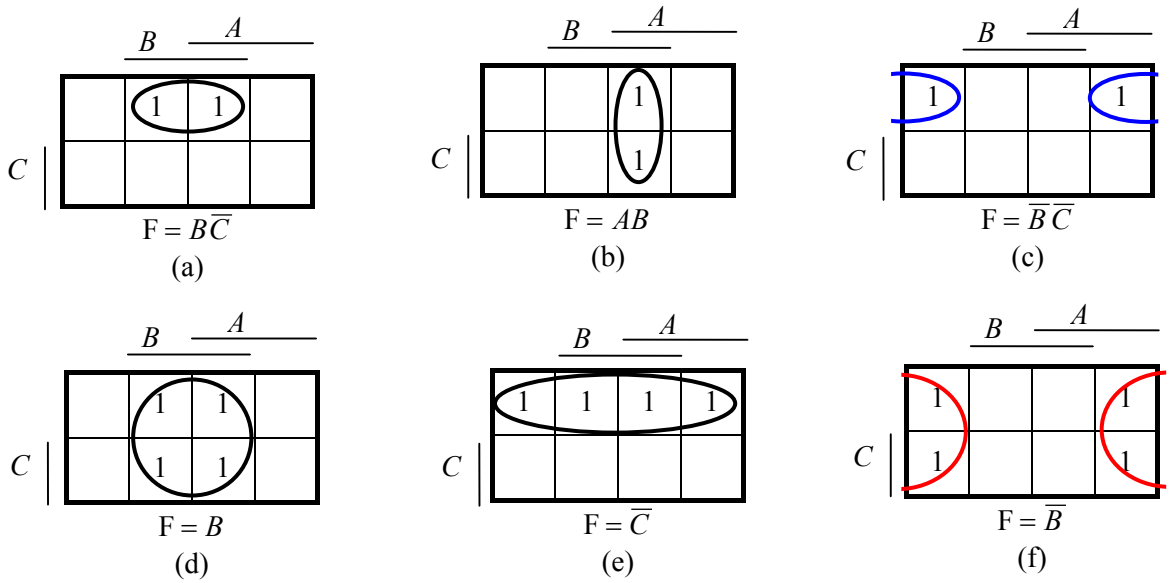


figure 2.16 : exemples de regroupements dans des diagrammes de Karnaugh à 3 variables

Dans le cas plus général où plusieurs regroupements sont possibles, il faut remarquer qu'une case peut être utilisée plusieurs fois, en raison de la propriété d'idempotence de la fonction OU : $A + A + ... = A$. Considérons les exemples de la figure 2.17 dans le cas de fonctions de 4 variables :

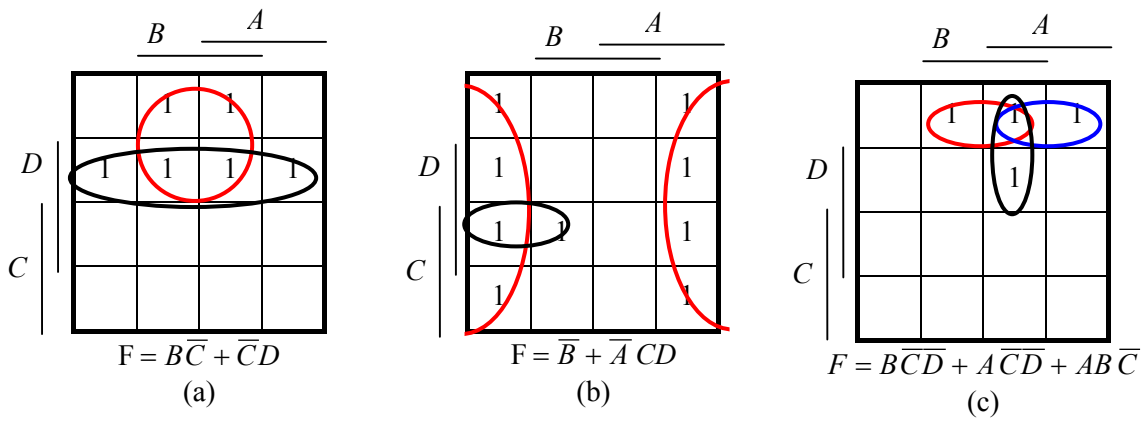


figure 2.17 : exemples de regroupements multiples dans des diagrammes de Karnaugh de 4 variables

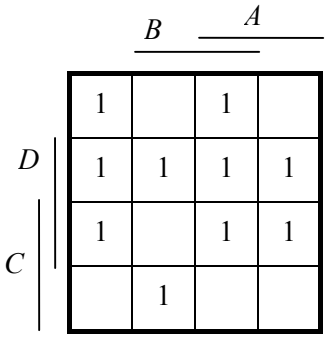
4.3.4.1 Technique à appliquer sur un diagramme de Karnaugh quelconque

Pour obtenir une expression simplifiée minimale, il faut inclure tous les 1 du tableau dans des groupements de taille 2^n en respectant les principes suivants :

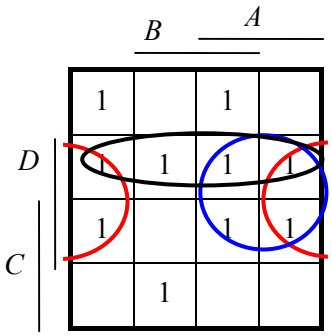
- Essayer de minimiser le nombre de groupements afin de minimiser le nombre de termes dans l'expression de la fonction. Il est alors préférable de rechercher les groupements en commençant par les cases qui ne peuvent se grouper que d'une seule façon. Ceci permet d'utiliser chaque 1 un minimum de fois.
- Vérifier que toutes les cases d'un groupe partagent le même nombre d'adjacences avec leurs congénères du groupe (soit *n* adjacences pour un groupe de 2^n cases).
- Les groupements de 1 doivent être les plus grands possibles (minimisation du nombre de variables).

4.3.4.2 Exemples

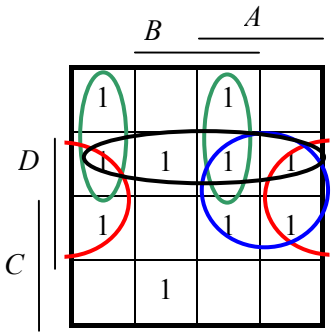
- **Exemple 1** : Simplification de
  $F_1 = \overline{A}\,\overline{B}\,\overline{C}D + AB\overline{C}D + \overline{A}\,\overline{B}\,C\overline{D} + \overline{A}\,BC\overline{D} + AB\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,CD + ABCD + A\overline{B}\,CD + \overline{A}\,BC\overline{D}$.



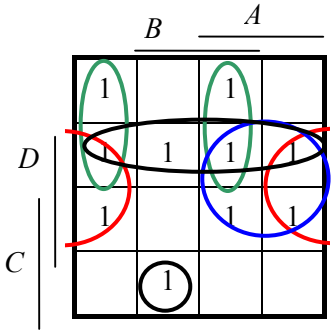
(a) diagramme de Karnaugh de F_1



(b) identification des groupes de taille 4



(c) identification des groupes de taille 2



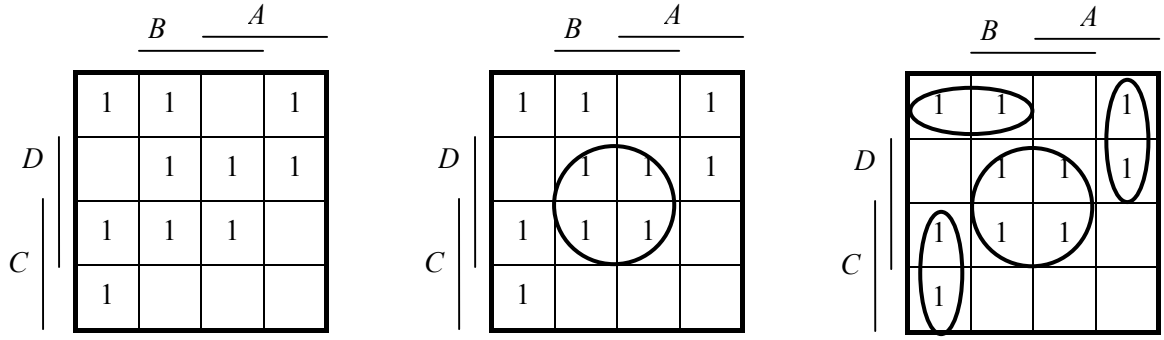
(d) identification des cases isolées

figure 2.18 : simplification de la fonction F_1

Après regroupement des 1 suivant les règles définies précédemment (figure 2.18), on obtient la forme simplifiée suivante : $F_1 = \overline{C}D + AD + \overline{B}D + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}\,BC\overline{D}$.

- **Exemple 2** : Simplification de

$F_2 = \overline{A}\,\overline{B}\,\overline{C}\overline{D} + \overline{A}\,B\overline{C}\overline{D} + A\overline{B}\,\overline{C}\overline{D} + \overline{A}\,B\overline{C}D + AB\,\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,CD + \overline{A}\,BCD + ABCD + \overline{A}\,\overline{B}\,C\overline{D}$.



(a) diagramme de Karnaugh de F_2 (b) identification des groupes de taille 4 (c) identification des groupes de taille 2

figure 2.19 : simplification de la fonction F_2

Les regroupements ont été effectués de façon à minimiser le nombre de groupes. On obtient alors $F_2 = BD + \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C$.

- **Exemple 3** : On va traiter ici un exemple de simplification à partir de la représentation de la fonction sous la seconde forme canonique. On place alors dans le tableau les 0 correspondants aux maxtermes intervenant dans l'expression de la fonction : $F_3 = (\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(A + \overline{B} + C)$.

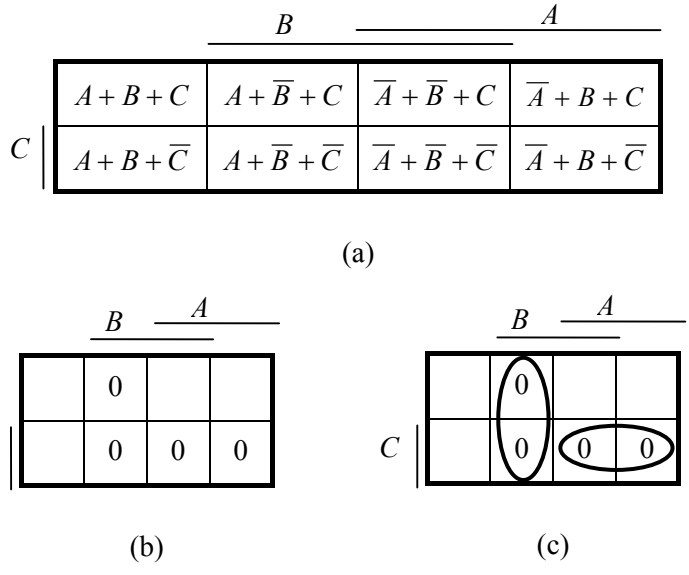
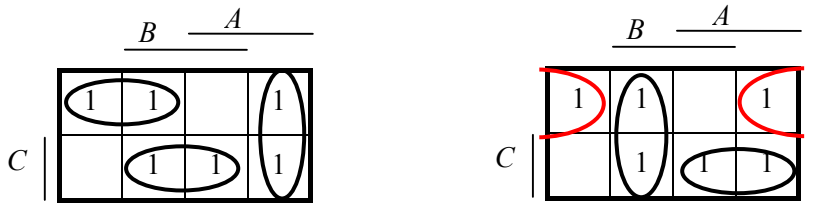


figure 2.20 : simplification de F_3 (représentée sous la deuxième forme canonique)
(a) position des maxtermes dans un diagramme de Karnaugh à 3 variables
(b) diagramme de la fonction F_3 : placement des 0
(c) résultat du regroupement des 0

On obtient, après simplification : $F_3 = (\overline{A} + \overline{C})(A + \overline{B})$.

Remarque : Le résultat de la simplification peut ne pas être unique. Par exemple, soit $F_4 = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\overline{C} + \overline{A}\,BC + ABC + A\overline{B}\,\overline{C} + A\overline{B}\,C$. La figure 2.21 donne deux résultats de simplification de même complexité.



(a) $F_4 = \overline{A}\,\overline{C} + BC + A\overline{B}$ (b) $F_4 = \overline{B}\,\overline{C} + \overline{A}\,B + AC$

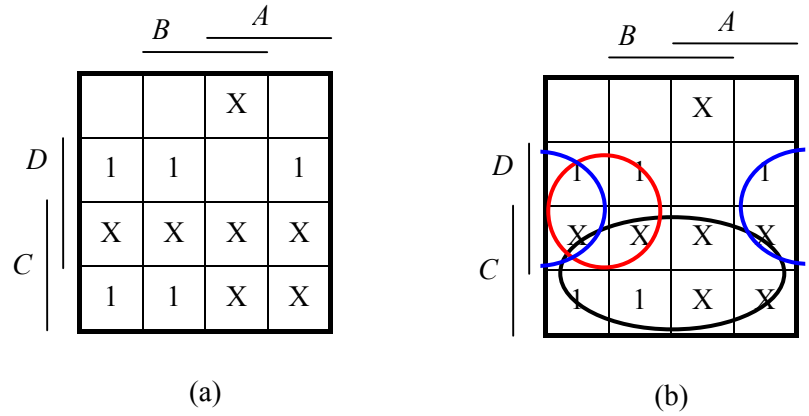
figure 2.21 : simplification de F_4

4.3.4.3 Cas des fonctions incomplètement spécifiées

Une fonction est dite **incomplètement spécifiée** si, pour certaines combinaisons des variables, elle prend indifféremment la valeur 0 ou la valeur 1, ou bien si l'occurrence de ces combinaisons n'est pas prévue dans la définition de la fonction. Dans ce cas, on a l'habitude d'inscrire un "X" ou un "-" dans les cases associées à ces combinaisons. On parle alors d'**états indifférents** ou **indéterminés** pour désigner les cases du diagramme correspondantes. Ces états peuvent être utilisés partiellement ou totalement pour simplifier la fonction. Les règles suivantes doivent alors être respectées :

- Effectuer d'abord les regroupements sans tenir compte des cases X ou -,
- Utiliser ensuite les cases X ou - pour réunir les groupes préexistants ou augmenter leur taille,
- Ne jamais utiliser une case X ou - pour créer un nouveau groupe, ceci irait à l'encontre du principe de minimisation du nombre de termes.

- **Exemple** : Soit la fonction F_5 définie par le diagramme de la figure 2.22(a) (forme ΣΠ).



(a) (b)

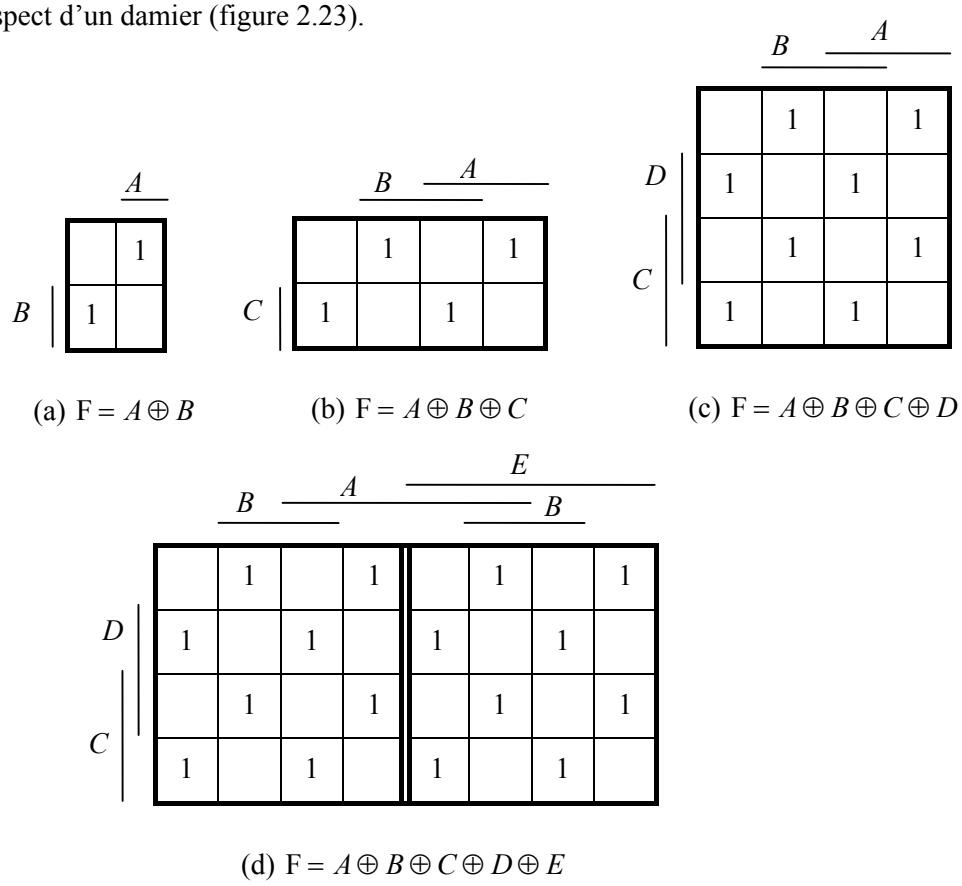
figure 2.22 : exemple de fonction incomplètement spécifiée

La présence d'états indifférents permet d'optimiser les regroupements comme indiqué sur la figure 2.22(b). Ainsi, pour l'écriture de F_5 sous forme simplifiée, les états indéterminés insérés dans

les groupements sont considérés comme des 1 et les autres comme des 0. On obtient alors $F_5 = C + \overline{A}\,D + \overline{B}\,D$.

4.3.4.4 Les cas du OU exclusif et du ET inclusif

L'expression booléenne $A \oplus B \oplus C \oplus ...$ vaut 1 si elle contient un nombre impair de 1. Si l'on complète le diagramme de Karnaugh correspondant, on observe que, en raison du codage de Gray, le tableau a l'aspect d'un damier (figure 2.23).



(a) F = $A \oplus B$ (b) F = $A \oplus B \oplus C$ (c) F = $A \oplus B \oplus C \oplus D$

(d) F = $A \oplus B \oplus C \oplus D \oplus E$

figure 2.23 : diagrammes de Karnaugh des opérateurs OU exclusif à 2, 3, 4 et 5 entrées

Pour obtenir les diagrammes de Karnaugh de la fonction ET inclusif, il faut échanger la place des 0 et des 1.

Dans les deux cas, la fonction n'est pas simplifiable sous la forme classique car aucun regroupement de 1 n'est possible. Il faut donc savoir reconnaître cet opérateur lorsqu'il apparaît dans un tableau de Karnaugh. Par exemple, la simplification de la fonction donnée par la figure 2.24 donne $F = A \oplus B \oplus C \oplus D + \overline{A}\,\overline{C} + BD$.

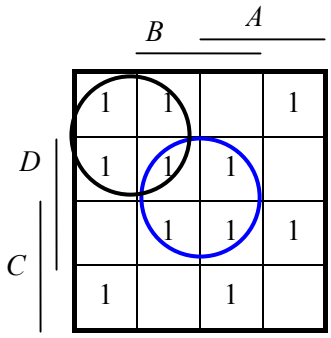


figure 2.24 : exemple de fonction contenant un OU exclusif

4.3.4.5 Conclusion

Lorsque le nombre de variables devient important, au-delà de 6, la manipulation des diagrammes de Karnaugh devient quasi-impossible. Il est alors nécessaire de recourir à des méthodes algorithmiques et d'utiliser un calculateur. Ce sont de telles méthodes qui sont utilisées dans les outils de synthèse automatique que l'on trouve actuellement sur le marché. Leur présentation sort du cadre de ce cours, mais les lecteurs intéressés pourront trouver de plus amples informations dans [Dan96] et [Sas93].

La minimisation des fonctions logiques permet une réalisation pratique utilisant un nombre minimal de composants, mais elle n'est pas une fin en soi. Dans les techniques actuelles d'intégration, la minimisation du nombre de composants n'est pas toujours le principal objectif : certaines contraintes de vitesse, de fiabilité peuvent même amener à augmenter la complexité d'un circuit. De plus, le progrès technologique aidant, la densité d'intégration est devenue aujourd'hui telle que le gain de quelques dizaines d'opérateurs logiques est souvent négligeable devant la complexité des circuits (plusieurs centaines de milliers d'opérateurs élémentaires par circuit en technologie CMOS).

5. Bibliographie

[Dan96] J. D. Daniels, *Digital design from zero to one*, John Wiley & Sons, 1996.

[GR87a] M. Gindre et D. Roux, *Electronique numérique : logique combinatoire et technologie, cours et exercices*, McGraw-Hill, Paris, 1987.

[LV86] J.-C. Lafont et J.-P. Vabre, *Cours et problèmes d'électronique numérique*, Ellipses, 1986.

[Sas93] T. Sasao, *Logic synthesis and optimization*, Kluwer Academic Publishers, 1993.

[TP94] J.-L. Danger, C. Degois, A. Galisson, J. Leroux, D. Roux, *Composants et fonctions de l'électronique numérique, cours*, polycopié Télécom Paris, 1994.

Chapitre 3 : Electronique des circuits logiques. Eléments de circuiterie MOS et CMOS

1. Bref historique de l'électronique et évolution des circuits intégrés

1.1 Le transistor bipolaire BJT (Bipolar Junction Transistor)

C'est en décembre 1947, dans les Bell Telephone Laboratories dirigés par Shockley, que Brattain et Bardeen ont mis pour la première fois en évidence l'effet transistor. Les premiers transistors fabriqués utilisaient un système de contacts à pointes (2 pointes de tungstène très rapprochées pour le collecteur et l'émetteur et une pastille de germanium pour la base). Ce système à double pointe métallique étant peu fiable, Shockley proposa en 1952 une simplification de fabrication avec le transistor à double jonction au germanium. S'ensuivirent rapidement les premières productions commerciales, à partir de 1953. En 1954, Texas Instruments lança la fabrication de transistors bipolaires au silicium permettant une plus forte dissipation calorifique que les transistors au germanium.

En 1956, Bardeen, Brattain, et Shockley reçurent le prix Nobel de physique pour l'invention du transistor. Ce fut le premier prix Nobel décerné pour un dispositif technique.

1.2 Les premiers circuits intégrés

En 1957, Siemens mit au point une technique de réalisation industrielle sur germanium, appelée technique **mesa**, destinée à augmenter la fiabilité, la fréquence de fonctionnement et à diminuer le prix de revient des transistors. Les jonctions étaient diffusées et les métallisations étaient déposées de manière globale sur une face d'une plaquette de germanium, puis les transistors étaient ensuite séparés les uns des autres.

En septembre 1958 naquit le premier **circuit intégré** : Kilby réalisa chez Texas Instruments un oscillateur sur une plaquette de 1,1 cm sur 0,5 cm comportant plusieurs transistors, condensateurs, et utilisant le procédé mesa sur silicium (cette contribution à l'invention du circuit intégré lui valut, plus de 40 ans plus tard, le prix Nobel de physique 2000 !). Cette même année vit la mise au point du procédé **planar** sur silicium (Hoerni et Noyce, chez Fairchild). Ce procédé reprend le concept de fabrication globale de plusieurs composants sur une même plaquette, mais les diffusions sont réalisées dans des zones délimitées par des masques. Le masquage utilise la couche de silice superficielle obtenue par simple oxydation du silicium et gravée par photogravure. L'oxyde de silicium sert également à passiver les transistors (c'est-à-dire les rendre étanche aux contaminations chimiques extérieures) et à isoler les connexions métalliques. Il n'existe pas de procédé planar sur germanium. Très rapidement (fin 1959), le procédé planar sur silicium s'imposa.

1.3 Le transistor à effet de champ FET (Field Effect Transistor)

Chronologiquement, le premier transistor à effet de champ a été mis au point après le transistor bipolaire. Cependant, en 1928, bien avant la fabrication du premier transistor bipolaire, un brevet déposé par un physicien américain, Lilienfeld, décrivait un dispositif à semi-conducteur possédant 3 électrodes et permettant de réaliser un poste récepteur avec des amplis HF et BF. Ce dispositif présentait la même structure et le même fonctionnement qu'un transistor FET. L'inventeur était cependant incapable de décrire convenablement le fonctionnement intime de son invention, les théories sur la matière solide n'étant alors pas assez avancées. Trop en avance sur son temps, l'invention fut reléguée et ne sortit de l'ombre que 35 ans plus tard ... après la mise au point du transistor MOS.

En 1951, Shockley proposa une structure de transistor à effet de champ à jonction JFET. Mais cette structure fut reléguée temporairement en raison de premiers essais de fabrication infructueux. Le premier JFET fut fabriqué en France par Tezner en 1958. Mais c'est finalement la mise au point de la technologie planar qui a permis l'émergence des transistors à effet de champ avec la découverte du transistor MOS (Métal-Oxyde-Semiconducteur) par Khang et Atalla dans les Bell Labs en 1960. Deux ans plus tard, Hofstein et Heiman (RCA, USA) déposèrent le premier brevet sur la fabrication intégrée des transistors MOS.

1.4 L'évolution des circuits intégrés depuis 1960

1.4.1 Évolution de la densité d'intégration

La course à l'intégration a débuté en 1960, avec le procédé planar. Ce sont surtout les circuits numériques qui ont tiré le développement des circuits intégrés, principalement pour satisfaire aux besoins de l'informatique.

- 1960 : quelques composants sur une plaquette,
- 1964 : circuits **SSI** (Small Scale Integration), moins de 100 transistors par puce,
- 1967 : circuits **MSI** (Medium Scale Integration), 100 à 3000 transistors par puce,
- 1973 : circuits **LSI** (Large Scale Integration), 3000 à 100 000 transistors par puce,
- 1982 : circuits **VLSI** (Very Large Scale Integration), 100 000 à 1 000 000 de transistors par puce,
- 1989 : circuits **ULSI** (Ultra Large Scale Integration), plus de 1 000 000 de transistors par puce.

Les années 60 furent l'ère de la fabrication de fonctions à l'aide de composants SSI et MSI (séries NS4000 : fonctions combinatoires élémentaires, bascules, petites unités arithmétiques et logiques, compteurs, ...) assemblés sur des petites cartes.

Les années 70 ont débuté avec l'invention de la mémoire RAM dynamique DRAM et la fabrication du premier microprocesseur 4 bits (Intel 4004) et se sont terminées avec la réalisation d'un microprocesseur 32 bits, le Motorola 68000. Les cartes étaient basées sur des circuits réalisant des fonctions standard complétées par des composants logiques SSI/MSI.

Dans les années 80, l'industrie du circuit intégré a été confrontée à la difficulté croissante de pouvoir identifier des fonctions standard adaptées à la conception d'une large gamme de systèmes. Est alors apparu le concept **ASIC** (Application Specific Integrated Circuit) ou **circuit spécifique** dont le principe est d'intégrer toutes les fonctions nécessaires à une application adonnée sur une seule puce. Par là-même, les circuits de type SSI/MSI ont alors quasiment disparu. Depuis la fin des années 80, une part de plus en plus importante du marché des ASICs est occupée par les circuits dits **programmables** ou **FPGA** (Field Programmable Gate Array), dont la fonctionnalité peut être programmée en laboratoire et qui sont utilisés pour les opérations de prototypage ou lorsque le nombre de puces à fabriquer est limité. Actuellement, des systèmes complets peuvent être embarqués sur une puce, pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue. On parle alors de System-on-Chip ou **SoC**.

En 1965, Gordon Moore (alors directeur de la recherche chez Fairchild, et plus tard co-fondateur d'Intel) notait dans un article de la revue *Electronics* que la complexité des composants intégrés de coût minimal avait doublé chaque année depuis le premier prototype fabriqué en 1959. Cette croissance exponentielle du nombre de transistors par circuits pour un coût minimal de fabrication a par la suite donné lieu à plusieurs reformulations et est devenue la « loi de Moore ». Dans les années 80, la loi de Moore prévoyait que le nombre de transistors par circuit serait doublé tous les 18 à 24 mois. Depuis le début des années 90, une interprétation courante est également la multiplication par deux de la puissance de calcul tous les 18 à 24 mois, pour un coût donné. Les différentes formulations de cette « pseudo-loi » ont ainsi permis de prévoir l'évolution des différentes familles de circuits intégrés depuis quarante ans. La
figure 3.1 illustre l'évolution de la complexité des microprocesseurs de la famille Intel depuis 45 ans.

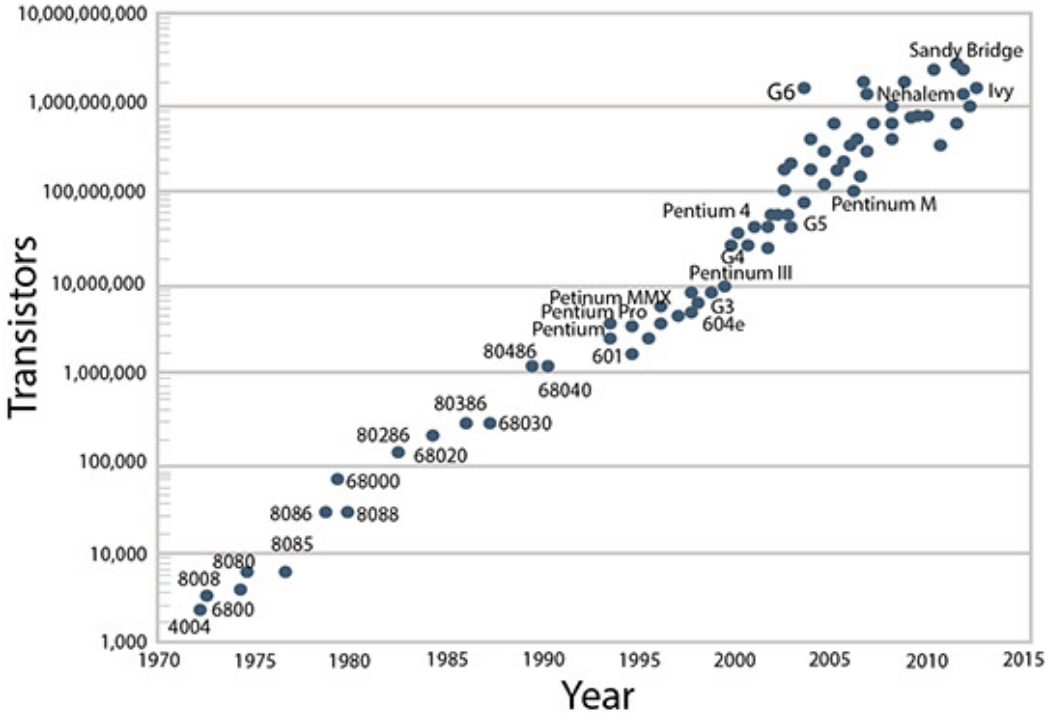


figure 3.1 : Illustration de la loi de Moore : évolution de la densité d'intégration des microprocesseurs Intel[1]

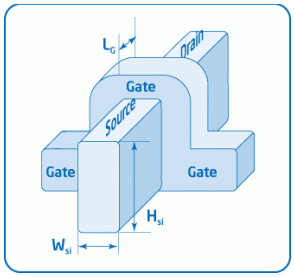
1.4.2 Évolution des circuits MOS

Le procédé MOS permet la réalisation de circuits à grande intégration, nécessitant de faibles puissances. Il est utilisé pour les mémoires et les microprocesseurs.

Quelques points marquants du développement des technologies MOS :

- 1966 : début du développement commercial des circuits intégrés MOS,
- 1971 : premier microprocesseur 4 bits (Intel 4004) en technologie MOS canal P à grille en aluminium, premières mémoires dynamiques de 1024 bits (Intel 1103),
- 1973 : microprocesseurs 8 bits (Intel 8008), en technologie NMOS canal N à grille en polysilicium, mémoires dynamiques de 4096 bits,
- 1974 : apparition de la logique CMOS à faible consommation, qui permet de réaliser des circuits de forte densité d'intégration,
- 1978-80 : mise au point de la technologie HMOS (High Speed CMOS) : microprocesseurs Intel 8086, Zilog Z8000, Motorola MC68000.
- Années 90 : mise au point de la technologie CMOS SOI (Silicon On Insulator). Insertion d'une couche d'isolant (oxyde de silicium) sous la zone active du substrat, qui permet une isolation diélectrique complète du transistor et entraîne une diminution des courants de fuite et des capacités de jonction.

[1] Source : University of Wisconsin-Madison



- Années 2010 : mise au point des transistors 3D. Dans le transistor 3D, la source et le drain ont une certaine épaisseur et la grille enserre ces conducteurs. Le contact se fait donc par les deux côtés et par le dessus, donc sur trois surfaces, au lieu d'une sur un transistor conventionnel.

1.4.3 Évolution des circuits bipolaires

La technologie bipolaire a moins rapidement évolué que la technologie MOS car elle ne permet pas une haute intégration.

Quelques points marquants du développement des technologies bipolaires :

- 1974 : utilisation des logiques TTL (Transistor-Transistor Logic) et TTL Schottky (TTL avec diodes Schottky pour augmenter la rapidité) pour des petits montages logiques rapides,
- 1975 : technologie I^2L (Integrated Injection Logic) utilisant des transistors NPN à multicollecteurs et présentant une plus forte densité d'intégration que le TTL,
- 1979 : technologie ECL (Emitter Coupled Logic), permettant de réaliser des circuits très rapides.

1.4.4 Les technologies alternatives

Au début des années 70, un nouveau composant aux performances intéressantes a fait son apparition. Il s'agit d'un transistor à effet de champ à jonction métal-semiconducteur (**MESFET**) sur substrat en **arséniure de gallium** ou **GaAs**. Ce matériau présente des propriétés électriques prometteuses :

- la mobilité des porteurs de type N est 6 fois supérieure à celle du silicium, et la vitesse limite des porteurs est plus élevée que dans le silicium, ce qui permet de réaliser des composants à faibles temps de propagation,
- le matériau non dopé présente une haute résistivité qui constitue un isolant naturel des composants intégrés.

Les techniques d'intégration sur GaAs ont commencé à être maîtrisées au milieu des années 80. Ce type de transistor a paru alors bien adapté à la réalisation de circuits intégrés numériques rapides (fréquences de fonctionnement au-delà du GHz). L'arséniure de gallium a cependant des points faibles vis à vis du silicium, essentiellement :

- l'absence d'oxyde stable compliquant les procédés planar (production 10 fois plus coûteuse que pour les circuits Si),
- la faible mobilité des trous rendant sans intérêt les logiques de type complémentaire analogues au CMOS.

Les circuits GaAs n'ont finalement pas réellement percé sur le marché des circuits intégrés numériques.

On observe ensuite, à la fin des années 80, l'apparition d'une technologie mixte, appelée **BiCMOS**, combinant des opérateurs CMOS et des transistors bipolaires, permettant ainsi d'atteindre

des performances en termes de vitesse, consommation, et densité d'intégration impossibles à obtenir avec chacune des technologies prise séparément. La technologie BiCMOS permet de réaliser des circuits plus rapides que les circuits CMOS, et avec une consommation plus faible que les circuits bipolaires. Dans les années 90, la technologie BiCMOS a connu une forte croissance, notamment pour la réalisation de circuits VLSI rapides : en 1997, les circuits BiCMOS occupaient 17 % du marché des circuits intégrés. Cependant, le coût élevé de production des circuits BiCMOS, lié à la complexité du process technologique, a fait rapidement régresser cette technologie au profit des circuits CMOS dont les performances en vitesse ne cessent de croître.

A partir du milieu des années 90, les technologies GaAs et BiCMOS sur silicium ont commencé à céder la place aux circuits **SiGe BiCMOS** qui combinent des transistors bipolaires à hétérojonction Si/SiGe aux opérateursCMOS pour permettre la fabrication de circuits numériques rapides (f_{max} au delà de10 GHz) et de faible consommation à plus faible coût.

1.4.5 Le marché des semi-conducteurs et des circuits intégrés[1]

Le marché mondial des semi-conducteurs est un marché à croissance moyenne fortement positive, plus de +10% en moyenne par an depuis vingt-cinq ans, mais qui présente d'importantes variations. A titre d'exemple, entre 1998 et 2000 les ventes de semi-conducteurs dopées par internet et les télécommunications mobiles ont atteint la croissance record de 60%. Celle-ci a ensuite été suivie d'une sévère baisse de plus de 30% en 2001 puis une stabilisation en 2002. Depuis 2003, le marché a globalement recommencé à croître. Les ventes se sont élevées à 333 G$ en 2014 avec une hausse de 9% par rapport à 2013 et les chiffres de 345 G$ et 355 G$ sont annoncés pour 2015 et 2016, soit respectivement +3,4 % et +3,1 % par rapport à l'année précédente. La répartition géographique des ventes en 2014 est la suivante : 58 % en Asie-Pacifique (Japon exclu), 20 % pour le continent américain, 11 % au Japon et 11 % en Europe. La répartition du marché en fonction du type de composants est décrite par le graphe de la figure 3.2. On observe que les circuits intégrés, numériques et analogiques, occupent plus des 3/4 du marché total.

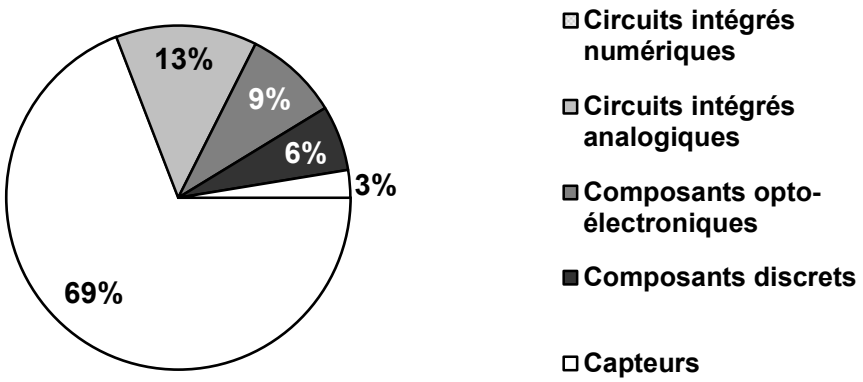


figure 3.2 : répartition du marché mondial des semi-conducteurs en 2014

[1] Source WSTS (World Semiconductor Trade Statistics) http://www.wsts.org/

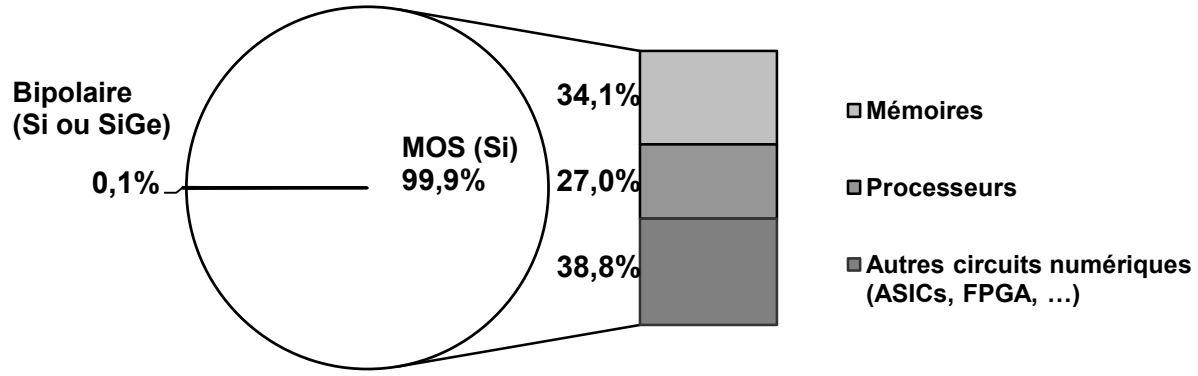


figure 3.3 : répartition du marché des circuits intégrés numériques en 2014

La figure 3.3 montre que les circuits numériques sont réalisés en quasi-totalité sur la base de technologies MOS. En pratique, hormis pour certaines mémoires faisant appel à des procédés de fabrication spécifiques (mémoires flash, cf. section 7 du chapitre 5, polycopié 2), ces circuits sont réalisés en technologie MOS complémentaire ou **CMOS**.

1.4.6 Perspectives d'évolution des technologies sur silicium

Le tableau 3.1 présente l'état actuel de la technologie de fabrication des circuits intégrés CMOS ainsi qu'un aperçu de son évolution dans les années à venir.

	2013	**2015**	**2017**	**2021**	**2025**	**2028**
DRAM/µproc. *half-pitches* (nm) (1/2 distance min. entre 2 gravures métalliques dans une cellule DRAM/ dans un circuit µprocesseur ou ASIC)	28/40	24/32	20/25	14/16	10/10	7,7/7
Longueur réelle du canal des transistors des µproc. et ASICs (nm)	20-23	17-19	14-16	10-11	7-8	5-6
Tension d'alimentation V_{DD} (V)	0,86	0,83	0,80	0,74	0,68	0,64
Nombre maximum de niveaux d'interconnexion	13	13	14	15	16	17
Densité d'intégration :						
DRAM (bits/chip)	4G	8G	8G	32G	32G	32G
Microprocesseurs (µP) et ASICs (Mtransistors/mm^2)	4	6	10	25	64	128
Fréquence maximale interne d'horloge des ASIC / µP (GHz)	5,5	5,9	6,4	7,5	8,8	9,9
	Actuel	Court terme			Long terme	

tableau 3.1: prévision de l'évolution de la technologie de réalisation des circuits intégrés jusqu'en 2028 [ITRS13]

2. Modèle du transistor MOS utilisé en électronique numérique

La suite de ce chapitre traite de la réalisation matérielle des circuits logiques à l'aide de transistors MOS exclusivement. Cette section est constituée d'une synthèse des connaissances concernant le transistor MOS nécessaires à la compréhension du fonctionnement des opérateurs logiques MOS et à l'estimation de leurs performances, en termes de surface, vitesse et consommation.

2.1 Rappels sur la structure du transistor MOS

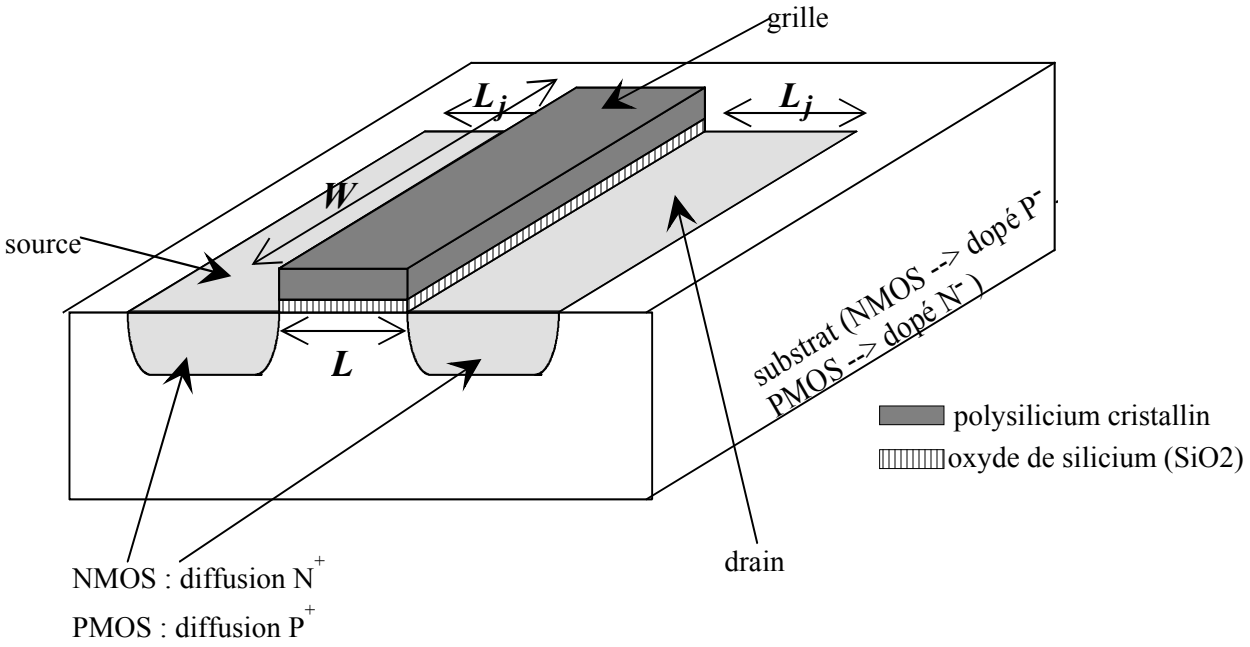


figure 3.4 : structure d'un transistor MOS

La figure 3.4 présente la structure simplifiée d'un transistor MOS. On appelle **surface active** du transistor la surface délimitée par le canal de conduction et qui correspond approximativement à la surface délimitée par l'oxyde de grille du transistor. Les quatre électrodes d'un transistor MOS, grille, source, drain, et substrat, sont dans la suite désignées par les lettres *G* (Gate), *S* (Source), *D* (Drain), et *B* (Bulk).

La figure 3.5 donne les symboles usuellement utilisés dans les schémas électriques pour représenter les transistors MOS canal N et canal P.

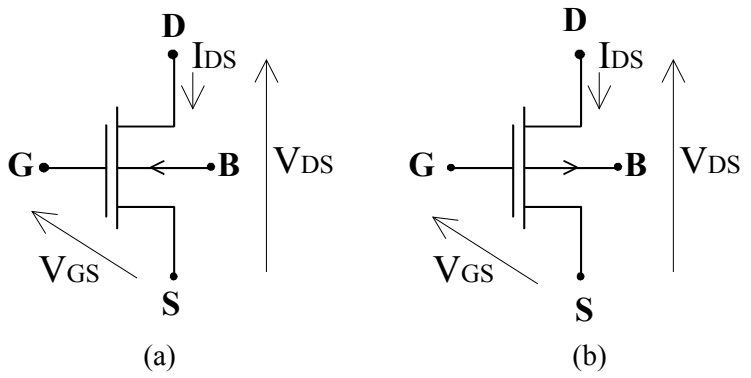


figure 3.5 : symboles électriques des transistors MOS canal N (a) et canal P (b)

2.2 Équations de conduction du transistor MOS

2.2.1 Transistor MOS canal N ou NMOS ($V_{DS} \geq 0$)

La conduction d'un transistor MOS est conditionnée par sa **tension de seuil** V_T (Threshold Voltage). Dans le cas d'un transistor NMOS :

- Si $V_{GS} < V_T$, $I_{DS} = 0$, le transistor est **bloqué**,
- Si $V_{GS} \geq V_T$, $I_{DS} \neq 0$, le transistor est **passant**.

Pour un transistor NMOS à **enrichissement**, $V_T > 0$, et pour un transistor à **déplétion,** $V_T < 0$.

On distingue, dans le cas où le transistor est passant, deux régimes de conduction :

- **Régime ohmique** (ou **linéaire**, ou **quadratique**) : $V_{DS} < V_{DSsat}$, où $V_{DSsat} = V_{GS} - V_T$.

$$I_{DS} = \beta\left[(V_{GS} - V_T) - \frac{V_{DS}}{2}\right]V_{DS}$$

avec $\beta = \mu_s C_{ox} \frac{W}{L}$, où

μ_s est la mobilité en surface des porteurs (électrons),

C_{ox} est la capacité de l'oxyde de grille par unité de surface,

W et L sont respectivement la largeur et la longueur du canal de conduction.

Cas particulier : **régime ohmique parfait**

Si $V_{DS} << V_{GS} - V_T$, $\quad I_{DS} \approx \beta(V_{GS} - V_T)V_{DS}$

- **Régime saturé** : $V_{DS} \geq V_{DSsat}$

$$I_{DS} = I_{DSsat}\left(1 + \frac{V_{DS} - V_{DSsat}}{V_e}\right)$$

où $I_{DSsat} = \frac{\beta}{2}(V_{GS} - V_T)^2$ et V_e est la tension d'Early ($V_e > 0$).

Pour la plupart des raisonnements, on pourra faire l'approximation $I_{DS} \approx I_{DSsat}$ en remarquant que $V_e >> V_{DS} - V_{DSsat}$.

La figure 3.6 représente un exemple de caractéristique courant/tension d'un transistor NMOS.

Pour un transistor NMOS passant, on définit la **résistance équivalente à l'origine** $R_{DS0}(V_{GS})$ par

$$R_{DS0}(V_{GS}) = \frac{1}{\left(\frac{\partial I_{DS}}{\partial V_{DS}}\right)_{V_{DS}=0}} = \frac{1}{\beta(V_{GS} - V_T)}$$

$R_{DS0}(V_{GS})$ est la résistance drain/source équivalente d'un transistor MOS en régime ohmique parfait.

Pour un transistor NMOS utilisé en commutation ($V_{GS} = V_{SS}$ ou V_{DD}), on désigne par R_{DS0} la résistance équivalente à l'origine pour $V_{GS} = V_{DD}$:

$$R_{DS0} = \frac{1}{\beta(V_{DD} - V_T)}$$

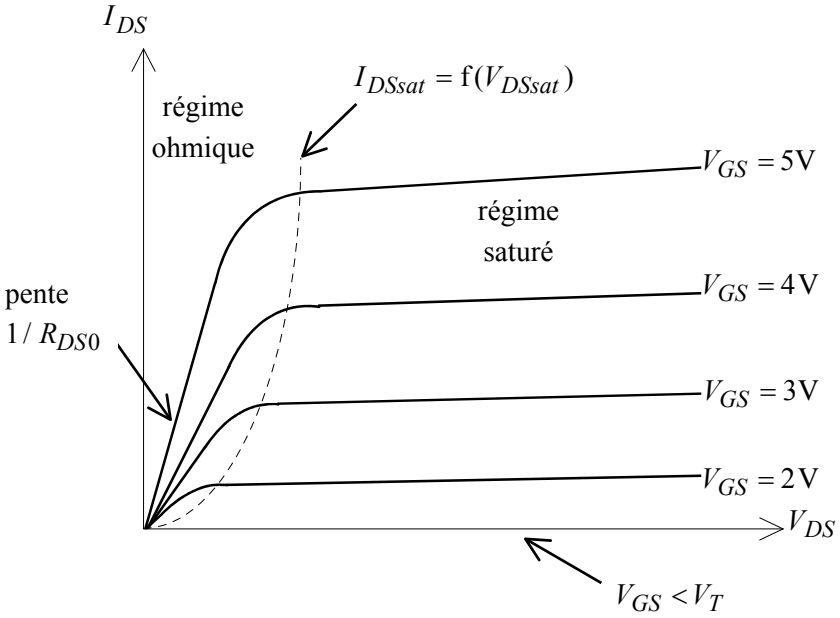


figure 3.6 : caractéristique courant/tension d'un transistor MOS à canal N

2.2.2 Transistor MOS canal P ou PMOS ($V_{DS} \leq 0$)

Dans le cas du transistor canal P, $V_T < 0$ pour un transistor à **enrichissement** et $V_T > 0$ pour un transistor à **déplétion**.

- Si $V_{GS} > V_T$, $I_{DS} = 0$, le transistor est **bloqué**,

– Si $V_{GS} \le V_T$, $I_{DS} \neq 0$, le transistor est **passant**.

Les équations de conduction sont alors les suivantes:

- **Régime ohmique** : $V_{DS} \ge V_{DSsat}$, où $V_{DSsat} = V_{GS} - V_T$

$$I_{DS} = -\beta\left[(V_{GS} - V_T) - \frac{V_{DS}}{2}\right]V_{DS}$$

Cas particulier : **régime ohmique parfait**

Si $|V_{DS}| << |V_{GS} - V_T|$, $\quad I_{DS} \approx -\beta(V_{GS} - V_T)V_{DS}$

- **Régime saturé** : $V_{DS} \le V_{DSsat}$

$$I_{DS} = I_{DSsat}\left(1 - \frac{V_{DS} - V_{DSsat}}{V_e}\right)$$

où $I_{DSsat} = -\frac{\beta}{2}(V_{GS} - V_T)^2$ et $V_e > 0$.

Dans la plupart des cas, on pourra également faire l'approximation $I_{DS} \approx I_{DSsat}$ en régime saturé.

La figure 3.7 représente un exemple de caractéristique courant/tension d'un transistor PMOS.

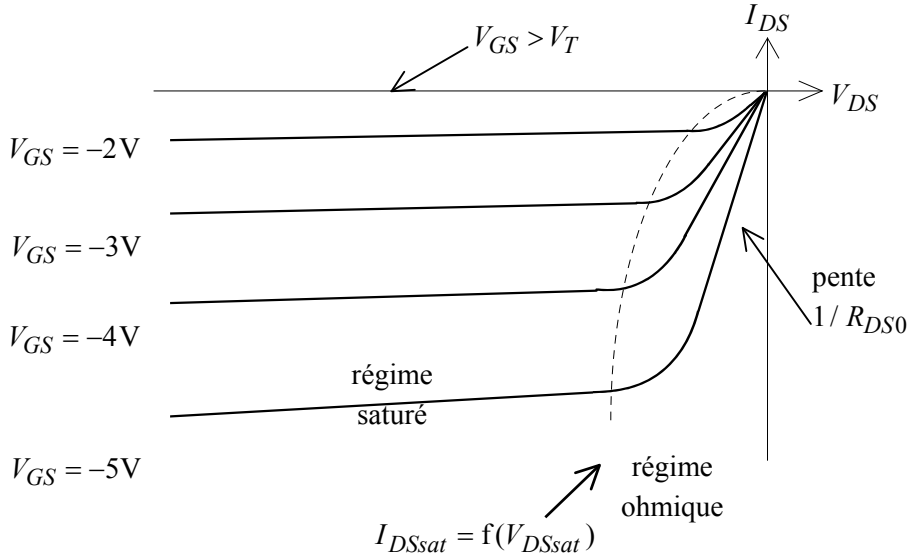


figure 3.7 : caractéristique courant/tension d'un transistor MOS canal P

Pour un transistor PMOS passant, la résistance équivalente à l'origine vaut $R_{DS0}(V_{GS}) = -\frac{1}{\beta(V_{GS} - V_T)}$. Pour un transistor PMOS utilisé en commutation ($V_{GS} = V_{SS}$ ou $-V_{DD}$), on désigne par R_{DS0} la résistance $R_{DS0}(V_{GS} = -V_{DD}) = \frac{1}{\beta(V_{DD} + V_T)}$.

2.3 Capacités parasites du transistor MOS

Cette section rappelle les principales capacités parasites du transistor MOS qui influent sur les performances dynamiques des opérateurs (temps de commutation).

2.3.1 Capacité de grille

- **Capacité intrinsèque grille/canal**

Cette capacité est due aux charges réparties le long du canal *lorsque le transistor est passant*. On a l'habitude de la représenter sous la forme de la somme de deux capacités : capacités grille/source C_{GS} et grille/drain C_{GD}.

En régime ohmique, la capacité totale grille/canal vaut WLC_{ox}. On considère alors, pour simplifier, que les charges présentes dans le canal proviennent pour moitié de la source et pour moitié du drain, soit :

$$C_{GD} \approx \frac{1}{2} WLC_{ox} \approx C_{GS} \text{ (régime ohmique)}$$

En régime saturé, on peut montrer que la capacité totale grille/canal vaut $\frac{2}{3} WLC_{ox}$. Le canal étant pincé du côté du drain, on considère alors que la capacité est entièrement due à la source :

$$C_{GD} = 0$$
$$C_{GS} \approx \frac{2}{3} WLC_{ox} \text{ (régime saturé)}$$

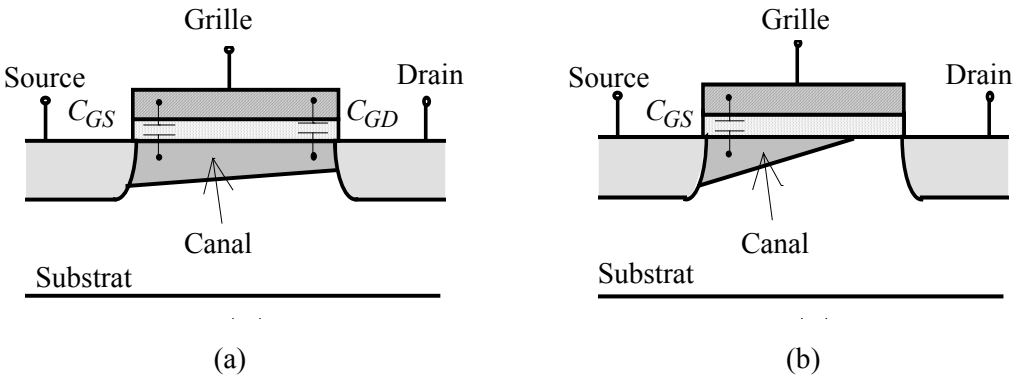


figure 3.8 : capacités intrinsèques du transistor MOS
(a) en régime ohmique, (b) en régime saturé

N. B. Les capacités de recouvrement de l'oxyde de grille sur le drain et la source sont négligées.

- **Capacité grille/substrat**

Cette capacité n'intervient que *lorsque le transistor est bloqué*. En effet, dans le cas contraire, le canal conducteur se comporte comme un écran électrique. Elle vaut

$$C_{GB} \approx WLC_{ox}$$

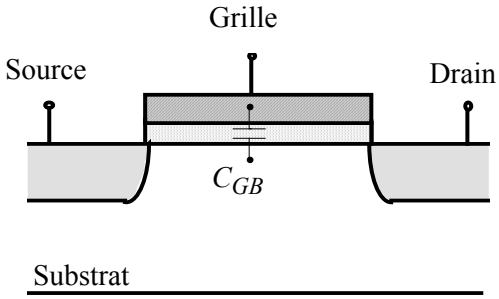


figure 3.9 : capacité grille/substrat d'un transistor MOS bloqué

- **Capacité totale de grille**

La capacité totale de grille est donc de l'ordre de grandeur de WLC_{ox}, quel que soit le régime de fonctionnement du transistor.

$$\boxed{C_G \approx WLC_{ox}}$$

2.3.2 Capacité des jonctions source/substrat C_{SB} et drain/substrat C_{DB}

Ce sont les capacités de transition des jonctions PN source/substrat et drain/substrat. Il est important de retenir qu'elles sont proportionnelles à la surface des jonctions, c'est-à-dire des zones de diffusion. En général, les surfaces de diffusion du drain et de la source sont identiques et C_{SB} et C_{DB} peuvent donc s'écrire sous la forme :

$$C_{SB} = C_{DB} = WL_jC_j$$

où C_j est la capacité de jonction par unité de surface.

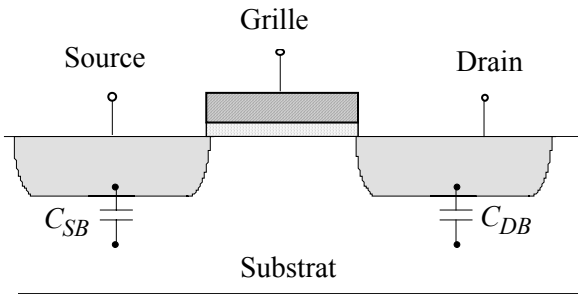


figure 3.10 : capacités de jonctions du transistor MOS

N. B.

Avec la mise au point récente des technologies « silicium sur isolant » SOI (Silicon-On-Insulator), les drains et sources des transistors sont diffusés dans une fine couche de silicium reposant sur un isolant (oxyde de silicium). Les jonctions PN drain/substrat et source/substrat, ainsi que les capacités parasites correspondantes, sont alors éliminées. Les premiers circuits CMOS sur substrat SOI ont été mis sur le marché en 1999.

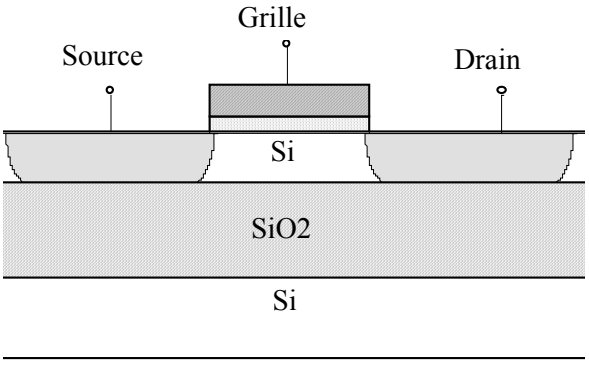


figure 3.11 : transistor MOS sur substrat SOI

2.3.3 Valeurs numériques

A titre d'illustration, pour une technologie CMOS 0,18 µm, les valeurs typiques des capacités sont les suivantes :

$$C_{ox} \approx 7{,}7 \times 10^{-3}\,\mathrm{pF}/\mu\mathrm{m}^2$$
$$C_j \approx 7{,}3 \times 10^{-4}\,\mathrm{pF}/\mu\mathrm{m}^2$$

Soit pour des transistors dont les dimensions sont

$$L \approx 0{,}18\ \mu\mathrm{m}$$
$$W \approx 0{,}33\ \mu\mathrm{m}$$
$$L_j \approx 0{,}33\ \mu\mathrm{m}$$

$$G_{GD} + C_{GS} \approx \begin{cases} 4{,}6 \times 10^{-4}\ \mathrm{pF\ en\ régime\ ohmique} \\ 3{,}0 \times 10^{-4}\ \mathrm{pF\ en\ régime\ saturé} \\ 0\ \mathrm{en\ régime\ bloqué} \end{cases}$$

$$C_{GB} \approx \begin{cases} 4{,}6 \times 10^{-4}\ \mathrm{pF\ en\ régime\ bloqué} \\ 0\ \mathrm{en\ régime\ ohmique\ \ ou\ saturé} \end{cases}$$

$$C_{SB} \approx 8 \times 10^{-5}\ \mathrm{pF}$$

3. Les circuits logiques MOS

Les **circuits logiques** ou **circuits numériques** sont des circuits électroniques réalisant des fonctions logiques. Les variables binaires à leurs entrées et sorties sont représentées par des niveaux de tension. Pour des raisons de performance et de compatibilité avec la logique TTL, l'excursion en tension pour les circuits MOS a longtemps été fixée entre V_{SS} = 0V et V_{DD} = 5V. Depuis quelques années, la valeur de la tension d'alimentation des circuits diminue à chaque saut technologique, afin de réduire la consommation. A titre d'exemple, les circuits intégrés réalisés en technologie CMOS 65 nm sont conçus pour fonctionner avec $V_{DD} \approx 1{,}1\text{V}$. Une tension proche de V_{SS} représente la valeur binaire 0 et une tension proche de V_{DD} la valeur binaire 1.

Dans la suite de ce chapitre, on désigne par **porte** ou **opérateur logique** un circuit logique réalisant une fontion logique élémentaire (NON, NAND, NOR, …).

Le premier circuit étudié est la porte logique d'inversion. C'est le circuit logique le plus simple. Son étude détaillée va permettre de définir un certain nombre de caractéristiques générales des portes logiques, représentatives de leurs performances.

3.1 Étude de l'inverseur NMOS à charge résistive

Considérons le montage de la figure 3.12. Le transistor utilisé est un transistor NMOS à enrichissement ($V_T > 0$).

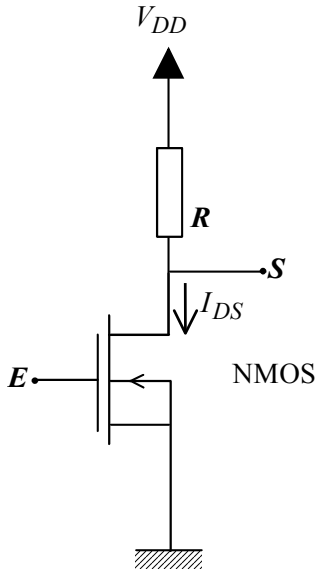


figure 3.12 : inverseur NMOS à charge résistive

3.1.1 Comportement logique de l'inverseur

L'utilisation d'un tel montage dans un circuit numérique suppose que les tensions appliquées sur l'entrée E prennent des valeurs proches de V_{SS} ou V_{DD}. Dans une première approche, le comportement du transistor peut alors être assimilé à celui d'un interrupteur commandé par sa tension de grille (figure 3.13).

Modèle du transistor NMOS utilisé en interrupteur :

- lorsque $E = V_{SS} < V_T$, le transistor est **bloqué** ($I_{DS} = 0$), il joue le rôle d'un interrupteur **ouvert**,
- lorsque $E = V_{DD} > V_T$, le transistor est **passant** ($I_{DS} \neq 0$), il joue le rôle d'un interrupteur **fermé**.

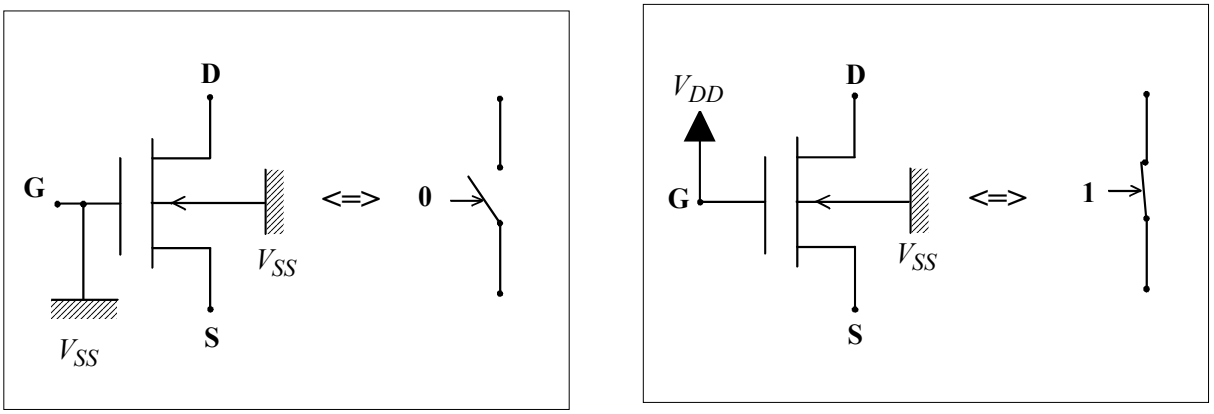


figure 3.13 : fonctionnement du transistor MOS à canal N en interrupteur

N.B. Rappel : Le substrat du transistor est relié à V_{SS} afin que les jonctions PN entre substrat et diffusions soient polarisées en inverse.

En utilisant ce modèle d'interrupteur, le comportement de l'inverseur est immédiat :

- Lorsque le transistor est bloqué, il n'y a pas de courant dans la résistance R et $S = V_{DD}$.
- Lorsque le transistor est passant, la sortie S est reliée à V_{SS}.

Le modèle de l'interrupteur idéal est suffisant pour déterminer la fonction logique d'un opérateur logique. Il n'est cependant pas assez précis pour estimer ses performances. Dans ce cas, l'utilisation du modèle fourni en section 2.2 est nécessaire. Le fonctionnement de l'inverseur est par la suite détaillé lorsqu'il est au repos (**caractéristiques statiques**), puis lors des **commutations** de l'entrée et de la sortie (**caractéristiques dynamiques**).

3.1.2 Caractéristiques statiques de l'inverseur

3.1.2.1 Niveaux de sortie

La superposition de la caractéristique du transistor et de la droite de charge d'équation $S = V_{DD} - RI_{DS}$ permet de déterminer les tensions de sortie de l'inverseur lorsque sont appliqués sur son entrée les niveaux V_{SS} et V_{DD} (figure 3.14).

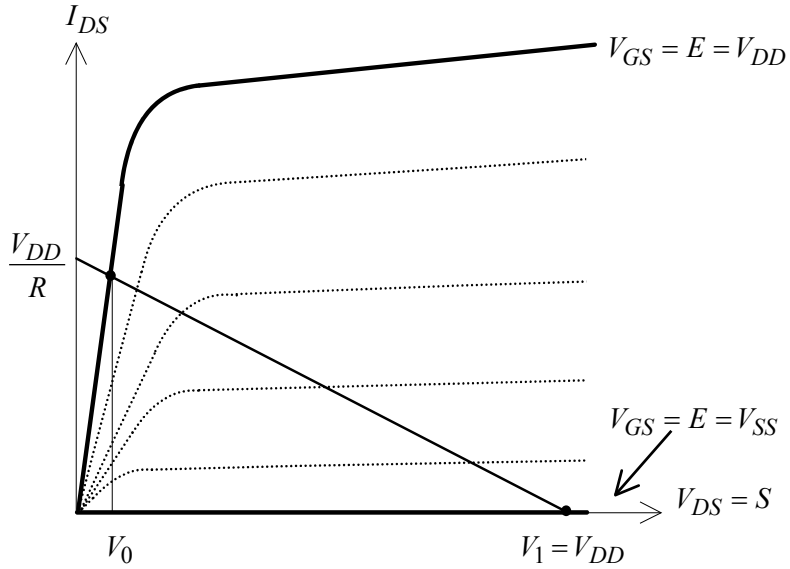


figure 3.14 : détermination des niveaux de sortie de l'inverseur

Lorsque $E = V_{GS} = V_{SS}$, en sortie $S = V_1 = V_{DD}$,

Lorsque $E = V_{GS} = V_{DD}$, on suppose que le transistor est en régime ohmique parfait (le niveau en sortie doit être proche de 0V pour être interprété comme un 0 logique), et une valeur approchée de $S = V_0$ est obtenue par résolution de :

$$\begin{cases} I_{DS} \approx \beta(V_{DD} - V_T)V_0 \\ V_0 = V_{DD} - RI_{DS} \end{cases}$$

soit

$$V_0 \approx \frac{V_{DD}}{1 + R\beta(V_{DD} - V_T)} \approx \frac{V_{DD}}{1 + \dfrac{R}{R_{DS0}}}$$

$$\boxed{V_0 \approx \frac{R_{DS0}}{R} V_{DD}}$$

car $R >> R_{DS0}$ (ordre de grandeur : $R \approx$ qqs 10 kΩ, $R_{DS0} \approx$ qqs 100 Ω).

Contrainte sur V_0 : il faut que le niveau bas de sortie soit inférieur à la tension de seuil V_T afin que l'opérateur attaqué par l'inverseur puisse fonctionner correctement.

A. N. Calcul de V_0 dans les conditions suivantes :

$V_{DD} = 5$ V, $V_{SS} = 0$ V
$\mu_s C_{ox} = 100\ \mu$A / V^2, $W / L = 3$
$V_T = 0{,}75$ V, $R = 10$ kΩ

On trouve $V_0 \approx 0{,}4$ V $< V_T$.

3.1.2.2 Caractéristique de transfert en tension et marges de bruit

La caractéristique de transfert de l'inverseur est obtenue en traçant $S = \mathrm{f}(E)$ pour V_{GS} variant entre V_{SS} et V_{DD} (figure 3.15).

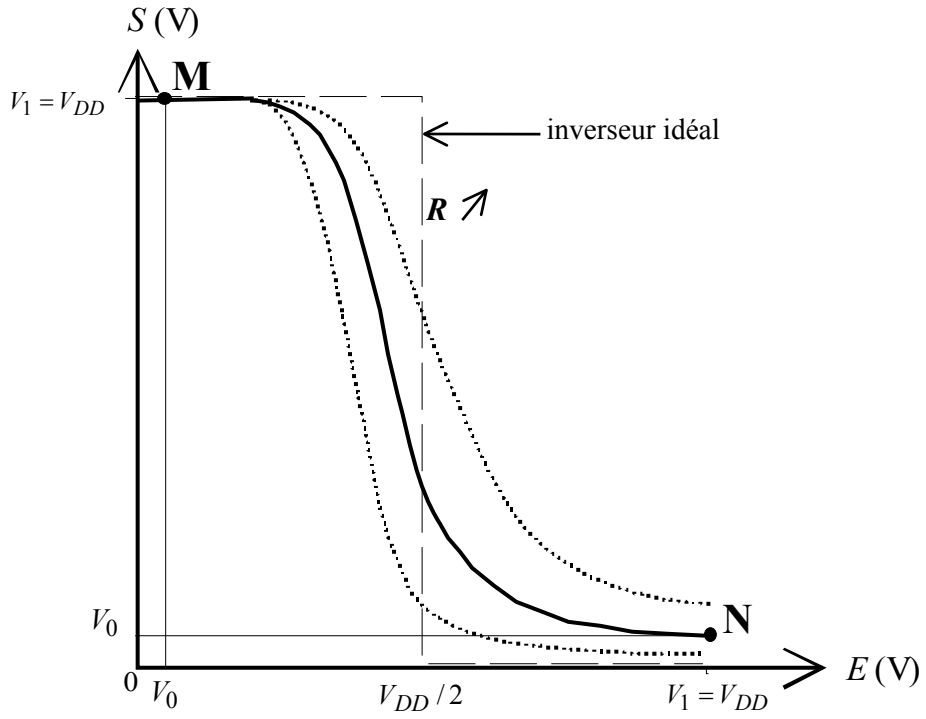


figure 3.15 : caractéristique de transfert de l'inverseur

Sur cette figure apparaît également, en pointillés larges, la caractéristique de transfert de l'inverseur idéal.

- **Points de fonctionnement**

Dans le cas de plusieurs inverseurs identiques connectés en chaîne, le 1 et le 0 logiques aux entrées et sorties des inverseurs internes à la chaîne sont représentés par les tensions $V_1 = V_{DD}$ et V_0 et correspondent aux points *M* ou *N* de la caractéristique de transfert. Ces points sont appelés **points de fonctionnement** de l'inverseur. Dans le cas de l'inverseur idéal, les coordonnées des points de fonctionnement sont (V_{DD}, V_{SS}) et (V_{SS}, V_{DD}). De ce point de vue, la caractéristique réelle se rapproche de la caractéristique idéale lorsque la valeur de *R* augmente car le niveau bas V_0 se rapproche alors de V_{SS}.

- **Points de gain unitaire et marges de bruit**

Les points *J* et *K* de la figure 3.16 sont les points de la caractéristique de transfert vérifiant $\frac{dS}{dE} = -1$ (**points de gain unitaire**). La zone *J K* de la caractéristique est appelée **région de transition**. Dans la région de transition, $\left|\frac{dS}{dE}\right| > 1$, et en dehors $\left|\frac{dS}{dE}\right| < 1$. Ainsi, si un signal dynamique parasite (bruit) vient perturber l'entrée de l'inverseur :

- si $E < E_J$ ou $E > E_K$, le bruit est transmis *atténué* à la sortie,
- si $E_J < E < E_K$, le bruit est transmis *amplifié* à la sortie, l'interprétation de la valeur logique en sortie de l'inverseur peut alors être erronée.

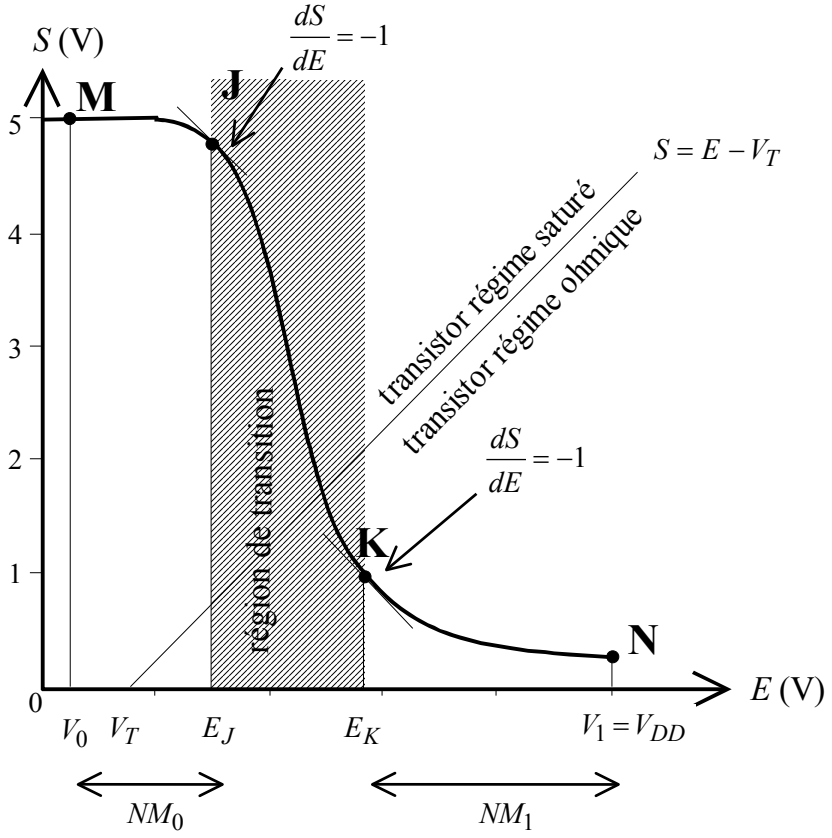


figure 3.16 : marges de bruits et région de transition

On appelle **marges de bruit** les plages de tension dans lesquelles E peut varier sans entraîner d'erreur sur l'interprétation du 0 ou du 1 logique en sortie de l'opérateur. On parle de **marge de bruit haute**, NM_1 (NM : Noise Margin) pour le niveau logique 1 et de **marge de bruit basse** NM_0 pour le niveau logique 0. Elles sont définies par :

$$\boxed{\begin{aligned} NM_1 &= V_1 - E_K \\ NM_0 &= E_J - V_0 \end{aligned}}$$

L'immunité au bruit d'un opérateur est d'autant meilleure que ses marges de bruit sont grandes.

La détermination des marges de bruit de l'inverseur NMOS à charge résistive passe par le calcul des tensions E_J et E_K. Elles sont obtenues par la résolution de l'équation $\frac{dS}{dE} = -1$, en régime saturé pour E_J, et en régime ohmique pour E_K, soit :

$$E_J = \frac{1}{\beta R} + V_T$$

$$E_K = \frac{1}{\beta R}\left[\frac{V_{DD}}{S_K} - 1\right] + V_T + \frac{S_K}{2} \text{ où } S_K = \sqrt{\frac{2V_{DD}}{3\beta R}}$$

A.N. Le calcul des marges de bruit dans les conditions suivantes :

$$V_{DD} = 5 \text{ V} \quad V_{SS} = 0 \text{ V}$$
$$\mu_s C_{ox} = 100\ \mu\text{A / V}^2,\ \text{W / L} = 3$$
$$V_T = 0{,}75 \text{ V},\ R = 10 \text{ k}\Omega$$

donne $E_J \approx 1{,}1$ V, $E_K \approx 2{,}5$ V, d'où $NM_0 \approx 0{,}7$ V et $NM_1 \approx 2{,}5$ V. Le niveau logique 1 présente une meilleure immunité au bruit que le niveau 0.

3.1.2.3 Consommation statique

La **puissance statique** consommée par un opérateur logique est définie par

$$\boxed{P_{stat} = \frac{1}{2}(P_0 + P_1)}$$

où P_0 est la puissance moyenne consommée lorsque la sortie de l'opérateur est à l'état bas et P_1 la puissance moyenne consommée à l'état haut. Cette définition suppose que la sortie de l'opérateur considéré est en moyenne aussi longtemps à l'état bas qu'à l'état haut.

Dans le cas de l'inverseur NMOS à charge résistive, $P_1 = 0$ car le transistor est bloqué lorsque $E = 0$, et $P_0 = V_{DD}\dfrac{V_{DD} - V_0}{R} \approx \dfrac{{V_{DD}}^2}{R}$, d'où

$$\boxed{P_{stat} \approx \frac{1}{2}\frac{V_{DD}^2}{R}}$$

A.N. Pour $V_{DD} = 5$ V et $R = 10$ kΩ, $P_{stat} = 1{,}25$ mW.

3.1.2.4 Influence de la valeur de **R** *sur les caractéristiques statiques de l'inverseur NMOS à charge résistive*

Du point de vue **statique**, l'inverseur est d'autant plus performant que ***R* est élevé :**

- la consommation est plus faible,
- la pente de la caractéristique de transfert dans la région de transition est plus abrupte (figure 3.15),
- le niveau de V_0 est plus proche du niveau nominal $V_{SS} = 0$ V.

3.1.3 Caractéristiques dynamiques

Chaque opérateur logique d'un circuit présente entre sa sortie et la masse de référence V_{SS} une charge capacitive C_L (*L* comme *Load*). Cette charge capacitive est constituée de 3 composantes principales :

- la capacité interne de sortie de l'opérateur (capacité drain/substrat, cf. 2.3.2),
- la capacité d'entrée pour les opérateurs en charge (capacité de grille, cf. 2.3.1),
- et une capacité due aux interconnexions entre l'opérateur et les opérateurs en charge (proportionnelle à la longueur des interconnexions).

Lorsque la sortie de l'opérateur considéré commute de l'état bas à l'état haut, ou *vice versa*, elle passe par un régime transitoire dont la durée depend de la valeur de la charge (figure 3.17).

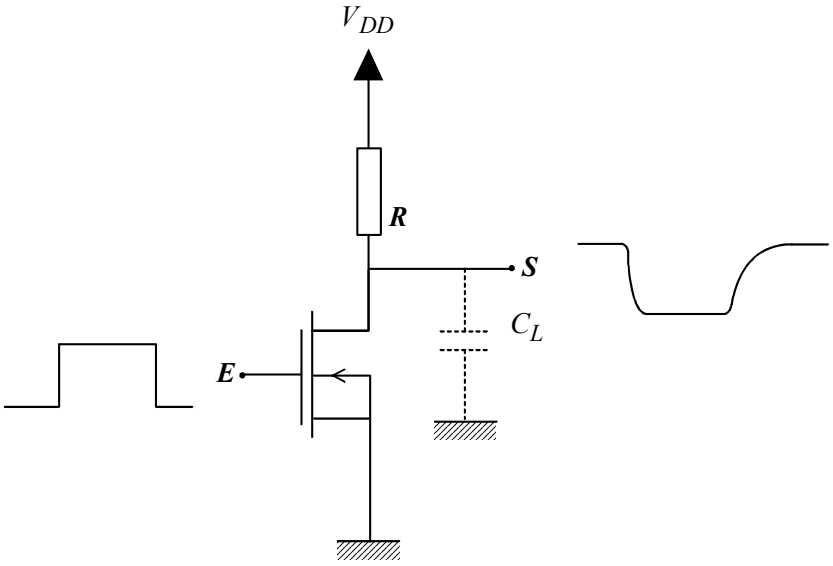


figure 3.17 : Réponse de l'inverseur NMOS à une impulsion

3.1.3.1 Temps de montée, de descente et temps de propagation

La rapidité de fonctionnement d'une porte logique dépend

- du temps de transition des signaux entre les 2 états logiques,
- du retard dû à la propagation de l'information à travers l'opérateur.

On définit le **temps de montée** t_r (respectivement **temps de descente** t_f) d'un signal comme étant le temps que met ce signal pour passer de 10% à 90% (respectivement de 90% à 10%) de la valeur de son amplitude totale (figure 3.18).

On appelle **temps de propagation** le retard de l'évolution d'une sortie par rapport à l'évolution d'une entrée de commande, les deux signaux étant considérés à la moitié de leur amplitude maximale (figure 3.18). On distingue le temps de propagation t_{pHL} correspondant au passage du signal de sortie

du niveau haut au niveau bas, et le temps de propagation t_{pLH} correspondant au passage du signal de sortie du niveau bas au niveau haut. Le **temps de propagation moyen** est alors défini par :

$$t_{p_{moy}} = \frac{t_{pHL} + t_{pLH}}{2}$$

Contrairement aux temps de montée et de descente, les temps de propagation s'additionnent le long d'une chaîne logique (association de plusieurs portes logiques).

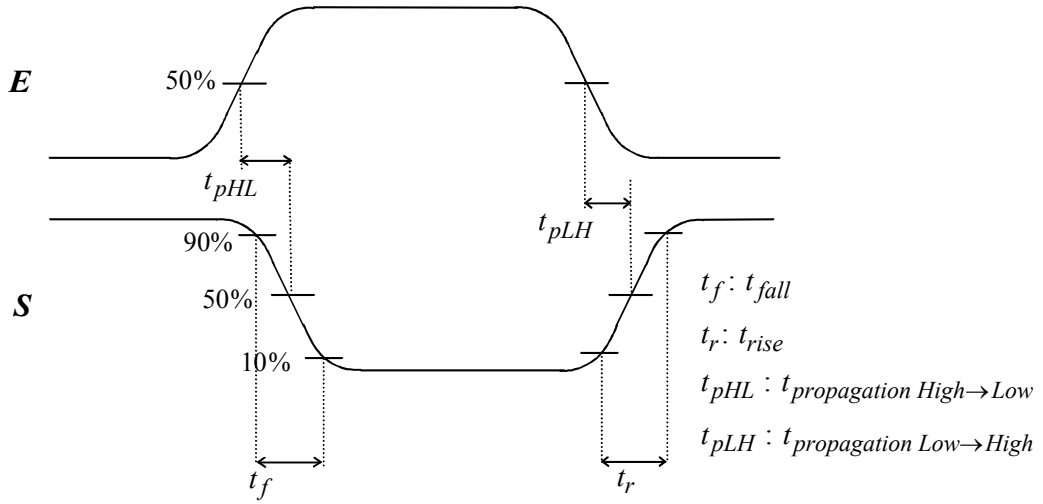


figure 3.18 : temps de montée, de descente, et de propagation

3.1.3.2 Calcul des temps de montée et de descente pour l'inverseur NMOS à charge résistive. Résistance équivalente d'un transistor NMOS

- **Temps de montée**

On suppose qu'au départ l'inverseur est au repos, E est à l'état logique 1, la capacité C_L est déchargée, la sortie S est à l'état logique 0 ($S = V_0$). Lorsque E passe à la valeur logique 0, le transistor se bloque instantanément, et la capacité C_L se charge à travers la résistance R jusqu'à ce que $S = V_1 = V_{DD}$.

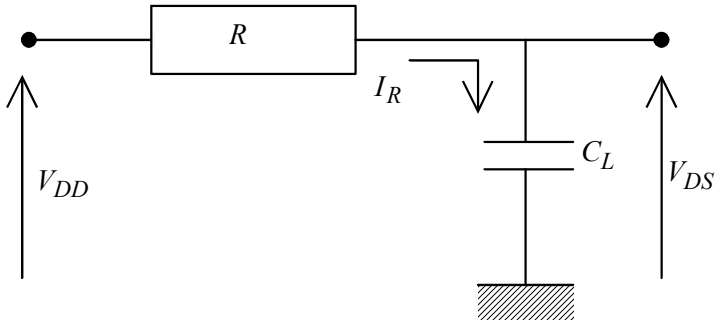


figure 3.19 : schéma équivalent de l'inverseur NMOS pour le calcul du temps de montée

Le temps de montée de l'inverseur NMOS se calcule comme le temps de charge de la capacité C_L à travers la résistance R (figure 3.19).

On a

$$I_R = \frac{V_{DD} - V_{DS}}{R} = C_L \frac{dV_{DS}}{dt}$$

d'où

$$dt = RC_L \frac{dV_{DS}}{V_{DD} - V_{DS}}$$

et, en considérant que $V_0 << V_1 = V_{DD}$

$$t_r = RC_L \left(\int_{0,1V_{DD}}^{0,9V_{DD}} \frac{dV_{DS}}{V_{DD} - V_{DS}}\right) = -RC_L \left[\ln(V_{DD} - V_{DS})\right]_{0,1V_{DD}}^{0,9V_{DD}}$$

$$\boxed{t_r = \ln 9 RC_L \approx 2{,}2RC_L}.$$

A.N. Pour $R = 10\ \text{k}\Omega$, $C_L = 50\ \text{fF}$, $t_r \approx 1{,}1\ \text{ns}$

- **<u>Temps de descente</u>**

On suppose qu'au départ l'inverseur est au repos, E est à l'état logique 0, la capacité C_L est chargée, la sortie S est à l'état logique 1. Dans ces conditions, le transistor est bloqué et aucun courant ne circule dans le montage, la capacité est isolée. Lorsque E passe de 0 à 1, le transistor devient passant, un courant I_{DS} circule entre le drain et la source, qui provoque la décharge progressive de la capacité jusqu'à ce que $S = V_0$.

On suppose que les constantes de temps considérées dans cette transition sont très inférieures à RC_L. Le temps de descente de l'inverseur NMOS se calcule alors comme le temps de décharge de la capacité C_L à travers le transistor (figure 3.20), selon le même principe que précédemment, mais la relation $I = f(V_{DS})$ est plus complexe que dans le cas du réseau RC.

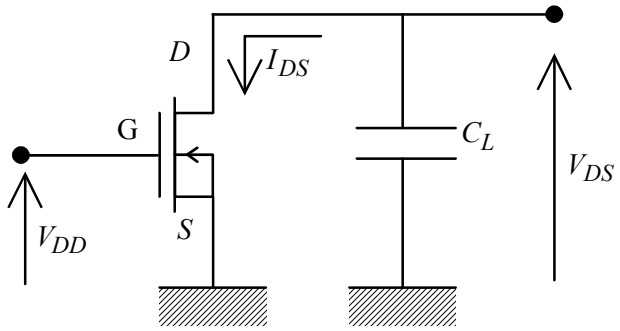


figure 3.20 : décharge de la capacité à travers le transistor

Au début de la décharge, le transistor est en régime saturé, puis il passe ensuite en régime ohmique. Il faut donc tenir compte des deux régimes pour le calcul de t_f. On utilisera les équations de

conduction simplifiées, c'est-à-dire que l'on suppose $V_e >> V_{DS} - V_{DSsat}$. On considérera également que $V_0 << V_1 = V_{DD}$ pour simplifier les calculs.

En début de décharge, lorsque $V_{DS} \geq V_{DSsat} \approx V_{DD} - V_T$

$$I_{DS} = \frac{\beta}{2}(V_{DD} - V_T)^2 = \frac{1}{2R_{DS0}}(V_{DD} - V_T)$$

or $I_{DS} = -C_L \frac{dV_{DS}}{dt}$,

d'où $dt = -2R_{DS0}C_L \frac{dV_{DS}}{V_{DD} - V_T}$.

En fin de décharge, lorsque $V_{DS} < V_{DD} - V_T$

$$I_{DS} = \beta\left[(V_{DD} - V_T) - \frac{V_{DS}}{2}\right]V_{DS} = \frac{1}{R_{DS0}}\left(1 - \frac{V_{DS}}{2(V_{DD} - V_T)}\right)V_{DS}$$

or $I_{DS} = -C_L \frac{dV_{DS}}{dt}$,

d'où $dt = -R_{DS0}C_L \frac{dV_{DS}}{V_{DS} - \frac{V_{DS}^2}{2(V_{DD} - V_T)}}$.

On en déduit

$$t_f = \underbrace{t_{fs}}_{\text{régime saturé}} + \underbrace{t_{fo}}_{\text{régime ohmique}}$$

avec $t_{fs} = \int_{0,9V_{DD}}^{V_{DD}-V_T} -2R_{DS0}C_L \frac{dV_{DS}}{V_{DD} - V_T} = \frac{2R_{DS0}C_L}{V_{DD} - V_T}\int_{V_{DD}-V_T}^{0,9V_{DD}} dV_{DS}$

et

$$t_{fo} = \int_{V_{DD}-V_T}^{0,1V_{DD}} -R_{DS0}C_L \frac{dV_{DS}}{V_{DS} - \frac{V_{DS}^2}{2(V_{DD} - V_T)}} = R_{DS0}C_L \int_{0,1V_{DD}}^{V_{DD}-V_T} \frac{dV_{DS}}{V_{DS} - \frac{V_{DS}^2}{2(V_{DD} - V_T)}}$$

Calcul de t_{fs} :

$$t_{fs} = \frac{2R_{DS0}C_L}{V_{DD} - V_T}\left[V_{DS}\right]_{V_{DD}-V_T}^{0,9V_{DD}} = 2R_{DS0}C_L \frac{V_T - 0,1V_{DD}}{V_{DD} - V_T}$$

Calcul de t_{fo} :

$$t_{fo} = R_{DS0}C_L \int_{0,1V_{DD}}^{V_{DD}-V_T} \left(\frac{1}{V_{DS}} + \frac{\frac{1}{2(V_{DD} - V_T)}}{1 - \frac{V_{DS}}{2(V_{DD} - V_T)}}\right) dV_{DS}$$

$$t_{fo} = R_{DS0}C_L\int_{0,1V_{DD}}^{V_{DD}-V_T}\left(\frac{1}{V_{DS}} - \frac{1}{V_{DS} - 2(V_{DD} - V_T)}\right)dV_{DS}$$

$$t_{fo} = R_{DS0}C_L\Big[\ln V_{DS} - \ln\left|V_{DS} - 2(V_{DD} - V_T)\right|\Big]_{0,1V_{DD}}^{V_{DD}-V_T}$$

$$t_{fo} = R_{DS0}C_L\left[\ln\frac{V_{DS}}{2(V_{DD} - V_T) - V_{DS}}\right]_{0,1V_{DD}}^{V_{DD}-V_T}$$

$$t_{fo} = R_{DS0}C_L\ln\frac{19V_{DD} - 20V_T}{V_{DD}}$$

d'où

$$\boxed{t_f = R_{DS0}C_L\left[2\frac{V_T - 0,1V_{DD}}{V_{DD} - V_T} + \ln\frac{19V_{DD} - 20V_T}{V_{DD}}\right]}$$

A.N. Pour $V_{DD} = 5$ V, $V_T = 0,75$ V,

$$\boxed{t_f \approx 3R_{DS0}C_L}$$

Si l'on fait correspondre cette expression avec l'expression du temps de descente d'un réseau RC, $RC\ln 9$, on peut calculer la résistance équivalente du transistor NMOS passant :

$$t_f = \ln 9 R_N C_L$$

pour

$$\boxed{R_N \approx 1,3R_{DS0}}$$

R_N est donc la valeur de la résistance qui présenterait exactement le même temps de descente que le transistor.

N. B. R_N et R_{DS0} étant du même ordre de grandeur, il est fréquent d'utiliser R_{DS0} à la place de R_N.

A. N. Pour $\mu_s C_{ox} = 100\ \mu$A / V^2, $W / L = 3$, $C_L = 50$ fF,

$$R_{DS0} = \frac{1}{\beta(V_{DD} - V_{T0})} \approx 780\ \Omega$$

$$R_N \approx 1\ \text{k}\Omega$$

$$t_f \approx 0,11\ \text{ns}$$

On remarque que $t_f << t_r$.

- **Temps de propagation**

On peut calculer les temps de propagation t_{pLH} et t_{pHL} de manière similaire aux calculs précédents en intégrant entre 0 et $0{,}5V_{DD}$ pour t_{pLH} et entre V_{DD} et $0{,}5V_{DD}$ pour t_{pHL}.

On obtient alors

$$t_{pLH} = \ln 2 RC_L \approx 0{,}7RC_L$$

et

$$t_{pHL} = R_{DS0}C_L\left[\frac{2V_T}{V_{DD}-V_T} + \ln\frac{3V_{DD}-4V_T}{V_{DD}}\right]$$

A. N. Avec le mêmes valeurs numériques que précédemment,

$t_{pLH} \approx 0{,}35$ ns

$t_{pHL} \approx 1{,}2R_{DS0}C_L \approx 5\times 10^{-2}$ ns

$t_{p_{moy}} \approx 0{,}2$ ns

3.1.3.3 Consommation dynamique

On appelle **puissance dynamique** P_{dyn} consommée par une porte logique, la puissance dissipée par effet Joule lors de la charge et de la décharge de la capacité C_L, pendant les transitions. L'énergie fournie par l'alimentation lors de la charge de la capacité C_L (commutation *LH*) est égale à $C_L V_{DD}^2$. Dans le cas de l'inverseur NMOS à charge résistive, elle est répartie comme suit (cf. figure 3.21) :

- $\frac{1}{2}C_L V_{DD}^2$ est dissipée par effet Joule dans R,
- $\frac{1}{2}C_L V_{DD}^2$ est emmagasinée sous forme d'énergie potentielle par C_L.

Lors de la transition *HL*, l'énergie emmagasinée par C_L est dissipée dans le transistor.

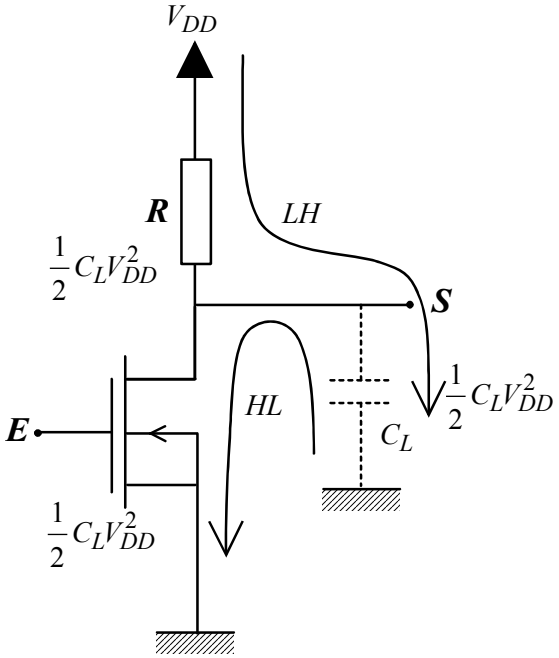


figure 3.21 : répartition de l'énergie lors des transitions *LH* et *HL*

Si on suppose que les commutations de la sortie de la porte logique se produisent en moyenne au rythme de la fréquence *f*, la puissance consommée en commutation vaut :

$$\boxed{P_{dyn} = fV_{DD}^2C_L}$$

A. N. Pour une fréquence de commutation $f = 10$ MHz, $P_{dyn} = 12{,}5\ \mu$W.

On définit la **puissance totale** comme étant la somme des puissances statique et dynamique :

$$\boxed{P_{tot} = P_{stat} + P_{dyn}}$$

Dans le cas de l'inverseur NMOS, $P_{dyn} << P_{stat}$ et $P_{tot} \approx P_{stat}$.

N. B. Le calcul précédent ne prend pas en compte le courant qui s'établit entre l'alimentation et la masse lors de la transition *HL* (lorsque $S < V_{DD}$, $I_R \neq 0$). La puissance ainsi dissipée est également négligeable devant P_{stat}.

3.1.3.4 Influence de R *sur les caractéristiques dynamiques de l'inverseur NMOS à charge résistive*

La valeur de *R* influe sur le temps de montée de l'inverseur : t_r **est directement proportionnel à *R***. Du point de vue **dynamique**, l'inverseur est donc d'autant plus performant (rapide) que ***R* est faible**. Ce résultat est en contradiction avec celui obtenu à la suite de l'analyse statique. Un compromis est donc à trouver entre performances statiques et performances dynamiques en fonction de l'application du circuit.

3.2 Étude de l'inverseur PMOS à charge résistive

Le montage de principe de l'inverseur PMOS est celui présenté en figure 3.22. La structure est duale de celle de l'inverseur NMOS.

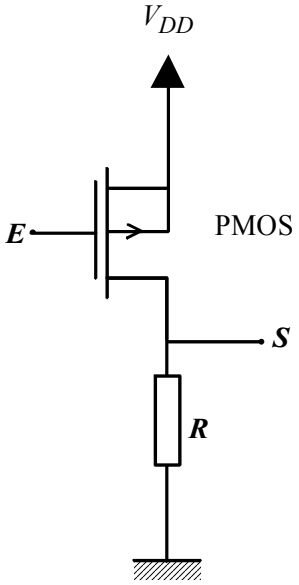


figure 3.22 : inverseur PMOS à charge résistive

Le transistor utilisé est un transistor à enrichissement ($V_T < 0$). D'un point de vue purement logique, son comportement est celui d'un interrupteur (figure 3. 23).

- Lorsque $E = V_{DD}$, $V_{GS} > V_T$, le transistor est **bloqué**, il joue le rôle d'un interrupteur **ouvert**,
- Lorsque $E = V_{SS}$, $V_{GS} < V_T$, le transistor est **passant**, il joue le rôle d'un interrupteur **fermé**.

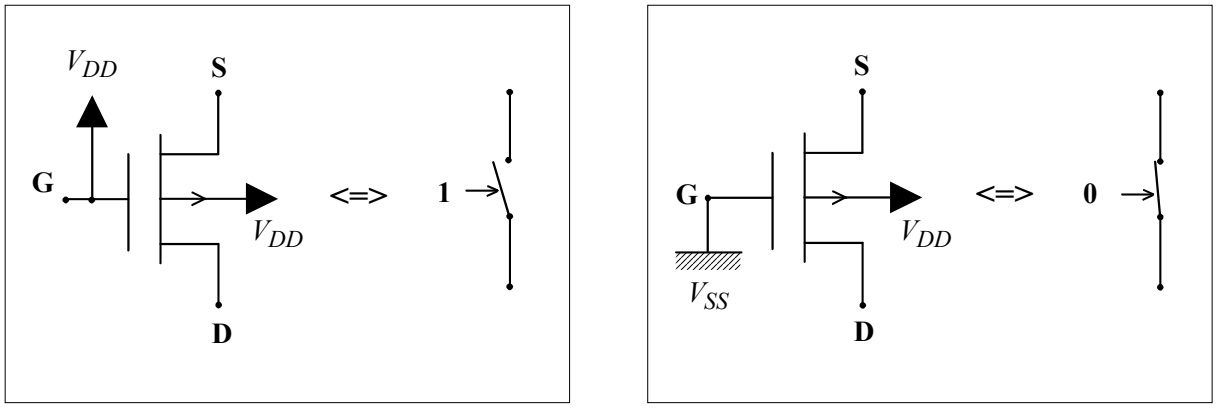


figure 3. 23 : fonctionnement du transistor MOS à canal P en interrupteur

N. B. Rappel : Le substrat du transistor est relié à V_{DD} afin que les jonctions PN diffusions/substrat soient polarisées en inverse.

L'étude des caractéristiques statiques et dynamiques de l'inverseur PMOS est similaire à celle menée pour l'inverseur NMOS. Pour résumer,

- **Points de fonctionnement**

Les niveaux de tension représentant le 0 et le 1 logique valent :

$$\boxed{V_0 = V_{SS}}$$

$$\boxed{V_1 \approx \frac{R}{R + R_{DS0}} V_{DD}}$$

- **Marges de bruit**

On observe que contrairement au cas de l'inverseur NMOS, la marge de bruit basse est plus grande que la marge de bruit haute. Le niveau logique 0 présente une meilleure immunité au bruit que le niveau logique 1.

$$NM_0 > NM_1$$

- **Consommation**

Globalement la consommation statique est la même que pour l'inverseur NMOS, mais elle a lieu lorsque la sortie de l'inverseur est à l'état haut.

$$P_0 = 0\ ,\ P_1 = V_{DD}\frac{V_1}{R} \approx \frac{V_{DD}^2}{R},$$

$$\boxed{P_{stat} \approx \frac{1}{2}\frac{V_{DD}^2}{R}}$$

La consommation dynamique est la même que pour l'inverseur NMOS.

$$\boxed{P_{dyn} \approx f V_{DD}^2 C_L}$$

- **Temps de commutation**

Le calcul des temps de montée et de descente t_r et t_f est similaire au calcul de t_f et t_r pour l'inverseur NMOS. On obtient, en considérant que $V_1 \approx V_{DD}$, $\boxed{t_r \approx 3R_{DS0}C_L}$, et $\boxed{t_f \approx 2{,}2RC_L}$. En pratique, $t_f >> t_r$.

En comparaison avec l'inverseur NMOS, l'inverseur PMOS présente des caractéristiques statiques et dynamiques qui sont duales de celles de l'inverseur NMOS.

Cependant, en pratique, les transistors PMOS ne sont pas utilisés seuls pour la réalisation de portes logiques. En effet, pour des transistors canal N et canal P issus du même processus de fabrication :

$$V_{TP} \approx -V_{TN}$$
$$\mu_{sN} \approx 3\mu_{sP}$$

soit, pour des transistors de mêmes dimensions, $R_{DS0P} \approx 3R_{DS0N}$. Dans ces conditions, on observe un rapport 3 entre t_{rP} et t_{fN} , autrement dit, l'inverseur NMOS est 3 fois plus rapide que l'inverseur PMOS.

3.3 Les inverseurs MOS réels

Un problème crucial se pose lors de l'intégration sur silicium de l'inverseur MOS : la réalisation de la résistance de charge *R* requiert une surface de silicium beaucoup plus importante que le transistor. Aussi, en pratique, le rôle de la charge est joué par un autre transistor.

A titre d'exemple, en technologie NMOS, une solution consiste à remplacer la résistance par un transistor NMOS à appauvrissement ou déplétion ($V_T < 0$), dont on relie la grille et la source (figure 3.24). Le transistor de charge est alors toujours passant. La figure 3.25 permet de comparer les caractéristiques de charge de l'inverseur à charge résistive et à charge déplétée dans des conditions de consommation statique identiques (même courant I_0). Les points de fonctionnement de l'inverseur à charge déplétée sont les mêmes que ceux de l'inverseur à charge résistive et ses caractéristiques dynamiques sont meilleures. En effet, le courant moyen parcourant la charge pendant les commutations est supérieur à celui parcourant la résistance (la courbe de charge est au dessus de la droite de charge de la résistance sur la figure 3.25). En conséquence, les charges et décharges de la capacité de sortie et donc les commutations sont plus rapides.

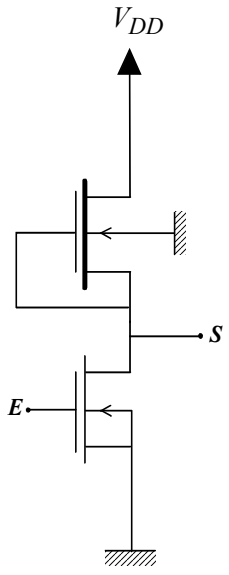


figure 3.24 : inverseur NMOS à charge déplétée

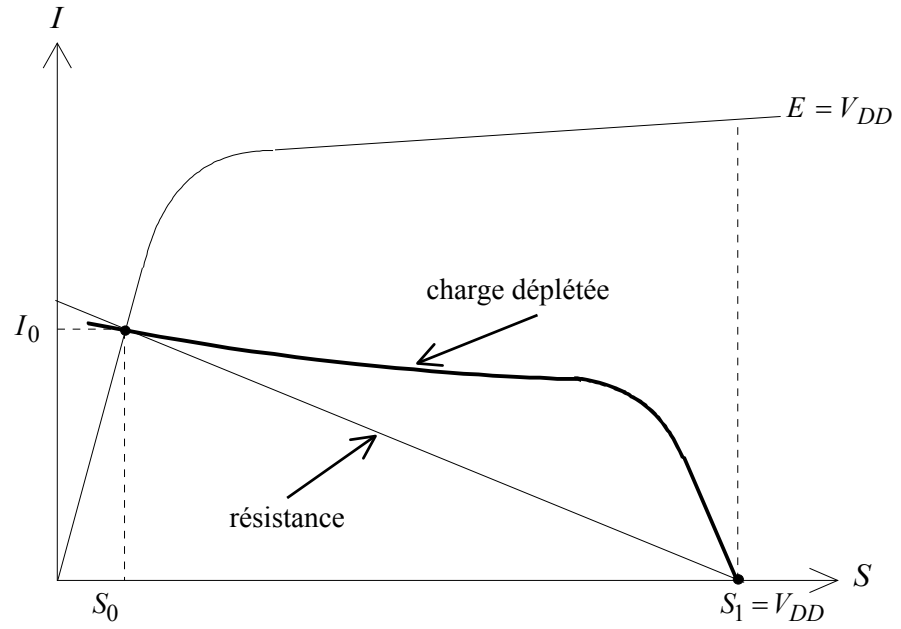


figure 3.25 : caractéristiques de charge des inverseurs à charges résistive et déplétée

En pratique, on ne fabrique presque plus de portes logiques NMOS (les circuits NMOS représentent en 2000 moins de 1% de l'ensemble des circuits MOS fabriqués). La solution la plus utilisée consiste à associer les deux types de transistors, à canal N et à canal P, afin d'éliminer leurs inconvénients et d'associer leurs avantages respectifs. Il s'agit de la **logique MOS complémentaire** ou **CMOS**. La structure de l'inverseur CMOS est obtenue par la mise en parallèle d'un inverseur NMOS et d'un inverseur PMOS dont on supprime les résistances de charge (figure 3.26).

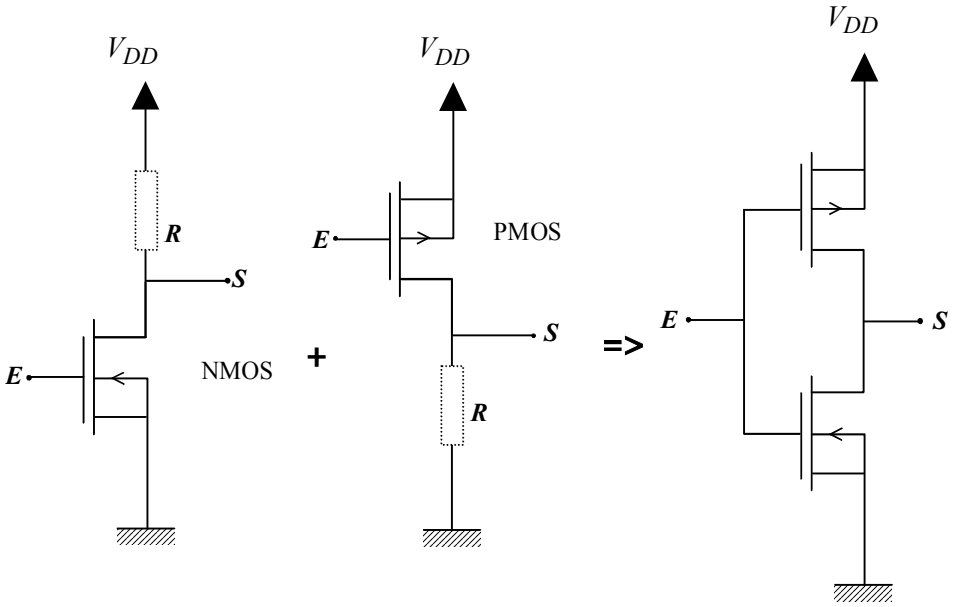


figure 3.26 : structure de l'inverseur CMOS

Chaque transistor joue à la fois le rôle de transistor de commande (fonctionnement en interrupteur) et de transistor de charge. On parle encore de structure à **charge active**. L'étude détaillée des caractéristiques statiques et dynamiques de l'inverseur CMOS fait l'objet de la section suivante.

3.4 Etude de l'inverseur CMOS

3.4.1 Comportement logique de l'inverseur

En utilisant le modèle d'interrupteur pour les 2 transistors, le comportement est le suivant :

- Lorsqu'un 1 logique est appliqué sur l'entrée de l'inverseur, le transistor canal P est bloqué, et le transistor canal N est passant, la sortie est reliée à V_{SS} (figure 3.27 (a)),
- Lorsqu'un 0 logique est appliqué sur l'entrée de l'inverseur, le transistor canal P est passant, et le transistor canal N est bloqué, la sortie est reliée à V_{DD} (figure 3.27 (b)).

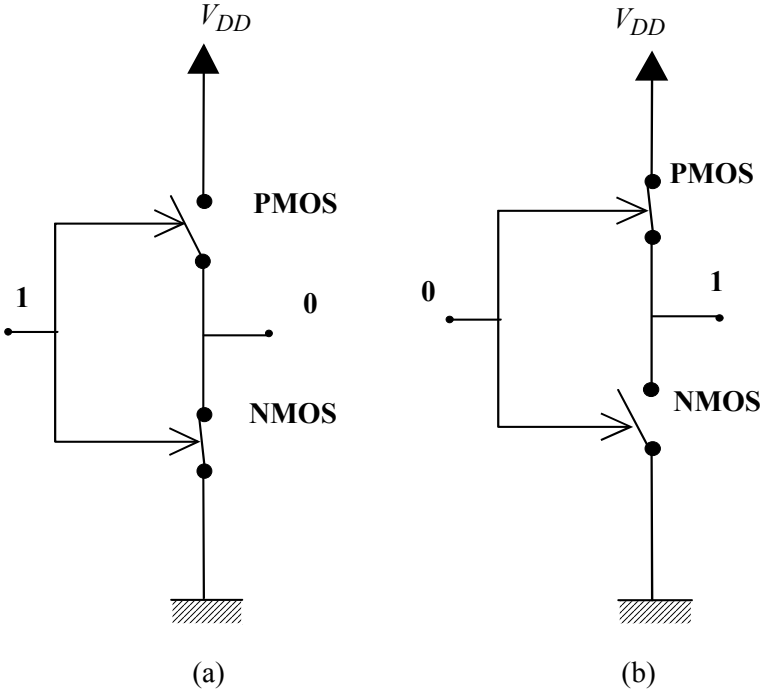


figure 3.27 : comportement logique de l'inverseur CMOS

3.4.2 Caractéristiques statiques de l'inverseur CMOS

3.4.2.1 Tracé de la caractéristique de transfert

Considérons l'inverseur CMOS de la figure 3.28, réalisé avec des transistors à enrichissement. Pour déterminer sa caractéristique de transfert de cet inverseur, on négligera l'effet Early : $V_{eN} \approx V_{eP} \approx \infty$.

Les conditions de conduction des transistors sont les suivantes :

- Transistor MOS canal N, T_N : $V_{GS} = E$, $V_{DS} = S$, $V_{DSsat} = V_{GS} - V_{TN} = E - V_{TN}$. Ce transistor est passant lorsque $E > V_{TN}$. Il est en régime saturé pour $S > E - V_{TN}$.

- Transistor MOS canal P, T_P : $V_{GS} = E - V_{DD}$, $V_{DS} = S - V_{DD}$, $V_{DSsat} = V_{GS} - V_{TP} = E - V_{DD} - V_{TP}$. Ce transistor est passant lorsque $E < V_{DD} + V_{TP}$. Il est en régime saturé pour $S < E - V_{TP}$.

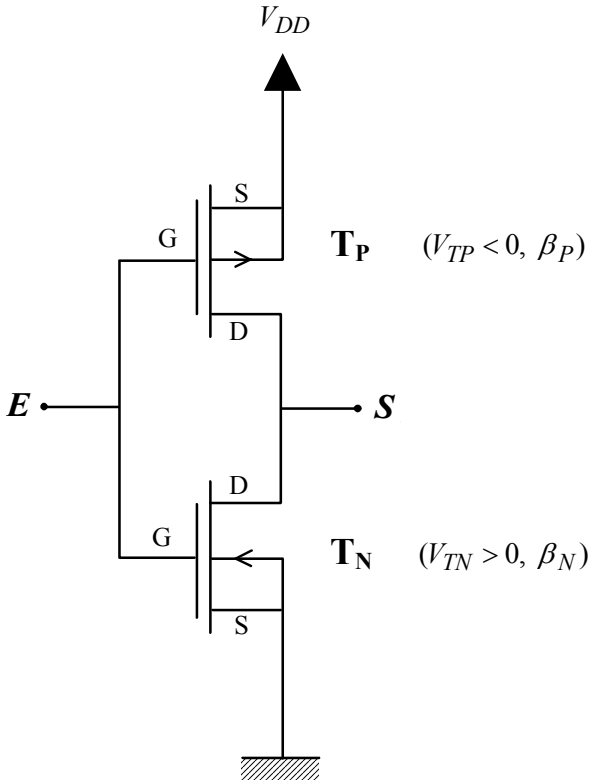


figure 3.28 : inverseur CMOS

La figure 3.29 résume les différents régimes de fonctionnement des deux transistors en fonction de *E* et *S*. Ce diagramme a été tracé dans le cas particulier d'un inverseur **électriquement symétrique**, à savoir :

$$\boxed{\begin{aligned} V_{TN} &= -V_{TP} = V_T \\ \beta_N &= \beta_P = \beta \end{aligned}}$$

Cette seconde relation suppose que, si les transistors sont issus du même processus de fabrication (même C_{ox}), $\mu_N \dfrac{W_N}{L_N} = \mu_P \dfrac{W_P}{L_P}$, soit $\boxed{\dfrac{W_P}{L_P} \approx 3\dfrac{W_N}{L_N}}$.

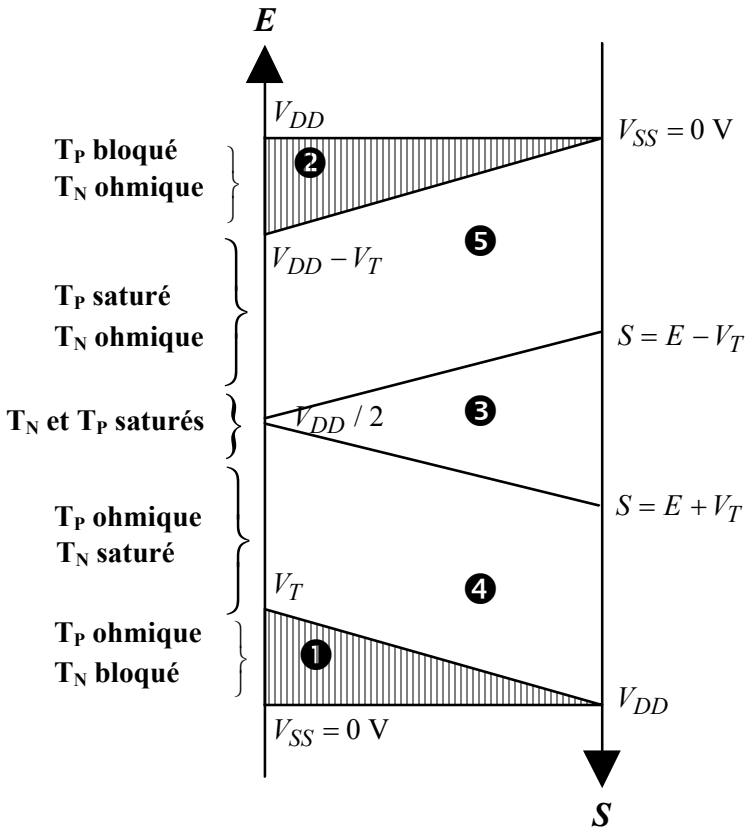


figure 3.29 : régimes de fonctionnement des transistors T_N et T_P en fonction de *E* et *S*

Ce schéma permet de définir 5 zones de la caractéristique de transfert de l'inverseur :

- zone ❶ : $E < V_T$, T_N est bloqué. Il ne passe pas de courant dans T_N, donc le courant dans T_P est également nul, bien que T_P soit passant. Donc $\boxed{S = V_{DD}}$.
- zone ❷ : $E > V_{DD} - V_T$, T_P est bloqué. Il ne passe pas de courant dans T_P, donc le courant dans T_N est également nul, bien que T_N soit passant. Donc $\boxed{S = V_{SS} = 0 \text{ V}}$.
- zone ❸ : Dans cette zone, les 2 transistors sont en régime saturé. On a $I_{DS_{T_N}} = -I_{DS_{T_P}}$, soit

$$\frac{\beta}{2}(E - V_T)^2 = \frac{\beta}{2}(E - V_{DD} + V_T)^2$$

d'où

$$E - V_T = \pm(E - V_{DD} + V_T).$$

La seule solution physiquement acceptable donne $E - V_T = -(E - V_{DD} + V_T)$, soit $\boxed{E = \frac{V_{DD}}{2}}$.

- zone ❹ T_N est en régime saturé et T_P en régime ohmique. L'égalité des courants dans les deux transistors s'écrit alors sous la forme :

$$I_{DS_{T_N}} = \frac{\beta}{2}(E - V_T)^2 = -I_{DS_{T_P}} = \beta\left[(E - V_{DD} + V_T) - \frac{1}{2}(S - V_{DD})\right](S - V_{DD})$$

La résolution de cette équation du 2nd degré donne :

$$\boxed{S = E + V_T + \sqrt{(V_{DD} - 2V_T)}\sqrt{(V_{DD} - 2E)}}$$

- zone ❺ T_P est en régime saturé et T_N en régime ohmique. L'égalité des courants dans les deux transistors s'écrit alors sous la forme :

$$I_{DS_{T_N}} = \beta\left[(E - V_T) - \frac{S}{2}\right]S = -I_{DS_{T_P}} = \frac{\beta}{2}(E - V_{DD} + V_T)^2$$

La résolution de l'équation donne dans ce cas :

$$\boxed{S = E - V_T - \sqrt{(V_{DD} - 2V_T)}\sqrt{(2E - V_{DD})}}$$

A partir de cette étude, le tracé de la caractéristique de transfert de l'inverseur CMOS est immédiat (figure 3.30).

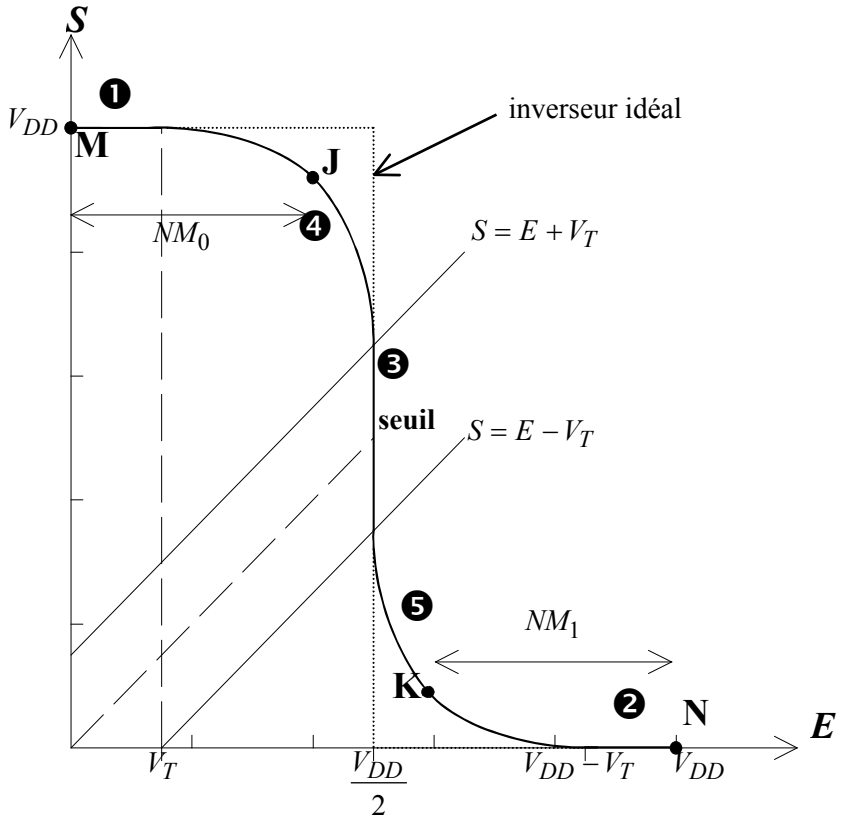


figure 3.30 : caractéristique de transfert de l'inverseur CMOS

La caractéristique ainsi obtenue est remarquablement proche de la caractéristique de transfert de l'inverseur idéal. En particulier, le seuil de commutation est situé à $E = \frac{V_{DD}}{2}$. En pratique, si les 2 transistors ne sont pas électriquement symétriques, notamment si $\beta_N \neq \beta_P$, le seuil de commutation est décalé par rapport à cette valeur. La pente au niveau du seuil est également nettement plus forte que dans le cas des inverseurs NMOS ou PMOS (même si elle n'est pas en pratique infinie, les modèles utilisés pour le calcul étant des modèles de conduction simplifiés !).

- **Points de fonctionnement**

Les points de fonctionnement *M* et *N* ont pour coordonnées (V_{SS}, V_{DD}) et (V_{DD}, V_{SS}), donc $V_0 = V_{SS}$ et $V_1 = V_{DD}$. Les niveaux haut et bas sont indépendants du R_{DS0}, donc du β des transistors. Cette propriété est vraie que l'inverseur soit électriquement symétrique ou non.

- **Marges de bruit**

Les marges de bruit sont données par

$$NM_0 = E_J - V_{SS} = E_J$$
$$NM_1 = V_{DD} - E_K$$

où *J* et *K* sont les points de gain unitaire ($\frac{dS}{dE} = -1$), et E_J et E_K leur abscisse sur la caractéristique de transfert.

Le point *J* appartient à la zone ❹ et le point *K* à la zone ❺. On dispose dans les 2 cas de la relation *S* = f (*E*). Il suffit donc de la dériver. On obtient :

$$E_J = \frac{3V_{DD} + 2V_T}{8}$$
$$E_K = \frac{5V_{DD} - 2V_T}{8}$$

d'où

$$\boxed{NM_0 = NM_1 = \frac{3V_{DD} + 2V_T}{8}}$$

A. N. Pour $V_{DD} = 1{,}5\ \text{V}, V_T = 0{,}5\ \text{V}, NM_0 = NM_1 \approx 0{,}7\ \text{V}$.

A la différence des inverseurs NMOS et PMOS, pour un inverseur CMOS électriquement symétrique, les 2 niveaux logiques présentent une immunité au bruit identique et proche de $\frac{V_{DD}}{2}$. En pratique, si les 2 transistors ne sont pas électriquement symétriques, les marges de bruit ne sont pas tout à fait égales.

3.4.2.2 Consommation statique

Lorsque l'inverseur est au repos, que S soit égal à V_{DD} ou V_{SS}, l'un des 2 transistors est bloqué, aucun courant ne passe dans l'inverseur. Les puissances moyennes consommées à l'état haut et à l'état bas, P_1 et P_0, sont donc nulles :

$$\boxed{P_{stat} = 0}$$

L'inverseur CMOS ne consomme pas en statique ! C'est une des propriétés majeures des circuits CMOS.

3.4.3 Caractéristiques dynamiques de l'inverseur CMOS

3.4.3.1 Consommation dynamique

En dynamique, l'alimentation fournit de l'énergie pour la charge de la capacité C_L, c'est à dire lors des transitions *LH*. L'énergie fournie est égale à $C_L V_{DD}^2$, comme pour l'inverseur NMOS (ou PMOS). Cette énergie est répartie comme suit :

- $\frac{1}{2} C_L V_{DD}^2$ est dissipée dans le transistor PMOS,
- $\frac{1}{2} C_L V_{DD}^2$ est emmagasinée par la capacité C_L.

Lors de la transition *HL*, l'énergie emmagasinée par C_L est dissipée dans le transistor NMOS.

La puissance totale consommée par un inverseur CMOS commutant à la fréquence *f* est donc égale à :

$$\boxed{P_{tot} = P_{dyn} = f V_{DD}^2 C_L}$$

N. B. Cette expression ne prend pas en compte le courant de court-circuit passant entre l'alimentation et la masse lorsque la tension d'entrée de l'inverseur se situe au voisinage du seuil de commutation, lorsque les deux transistors sont passants. Cette puissance de court-circuit vaut typiquement 10 % de $f V_{DD}^2 C_L$ et est par conséquent négligée.

3.4.3.2 Temps de montée, de descente et temps de propagation

Le calcul des temps de montée et de propagation *LH* de l'inverseur CMOS et de l'inverseur PMOS sont identiques. Il en va de même pour le calcul des temps de descente et de propagation *HL* des inverseurs CMOS et NMOS.

- **Temps de montée et temps de propagation *LH***

Lorsque E passe de V_{DD} à V_{SS}, le transistor T_N se bloque instantanément et la capacité se charge à travers le transistor T_P (figure 3.31). On obtient :

$$t_r = R_{DS0P} C_L \left[-2 \frac{V_{TP} + 0{,}1 V_{DD}}{V_{DD} + V_{TP}} + \ln \frac{19 V_{DD} + 20 V_{TP}}{V_{DD}} \right]$$

$$t_{pLH} = R_{DS0P} C_L \left[-\frac{2 V_{TP}}{V_{DD} + V_{TP}} + \ln \frac{3 V_{DD} + 4 V_{TP}}{V_{DD}} \right]$$

où

$$R_{DS0P} = \frac{1}{\beta_P (V_{DD} + V_{TP})} = \frac{L_P}{\mu_{Ps} C_{ox} W_P (V_{DD} + V_{TP})}$$

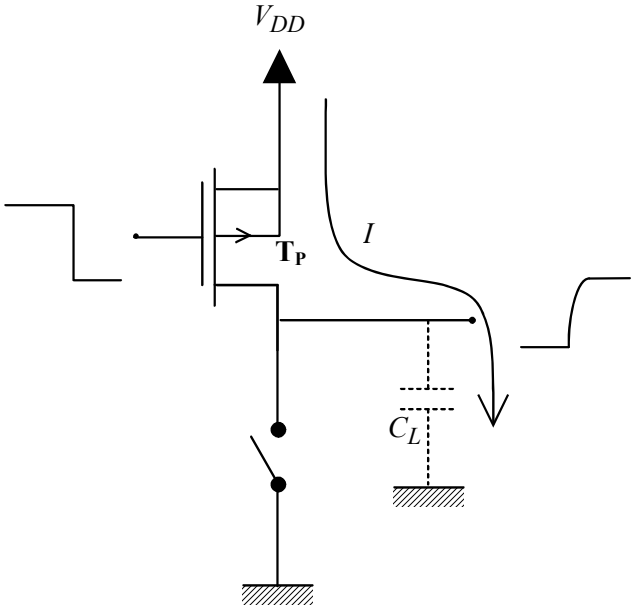


figure 3.31 : charge de C_L à travers T_P

Si $V_{DD} = 1{,}8$ V et $V_{TP} = -0{,}5$ V

$$\boxed{\begin{aligned} t_r &\approx 3 R_{DS0P} C_L \\ t_{pLH} &\approx 1{,}5 R_{DS0P} C_L \approx \frac{t_r}{2} \end{aligned}}$$

- **<u>Temps de descente et temps de propagation *HL*</u>**

Lorsque E passe de V_{SS} à V_{DD}, le transistor T_P se bloque instantanément et la capacité se décharge à travers le transistor T_N (figure 3.32). On obtient :

$$t_f = R_{DS0N} C_L \left[2 \frac{V_{TN} - 0{,}1 V_{DD}}{V_{DD} - V_{TN}} + \ln \frac{19 V_{DD} - 20 V_{TN}}{V_{DD}} \right]$$

$$t_{pHL} = R_{DS0N} C_L \left[\frac{2 V_{TN}}{V_{DD} - V_{TN}} + \ln \frac{3 V_{DD} - 4 V_{TN}}{V_{DD}} \right]$$

où

$$R_{DS0N} = \frac{1}{\beta_N (V_{DD} - V_{TN})} = \frac{L_N}{\mu_{Ns} C_{ox} W_N (V_{DD} - V_{TN})}$$

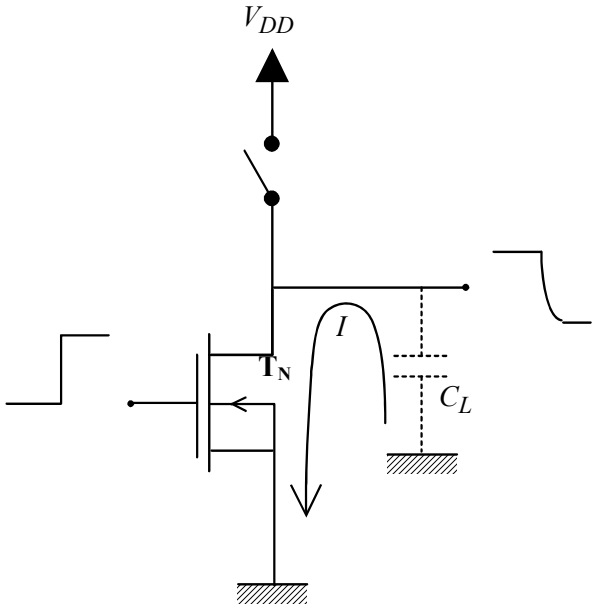


figure 3.32 : décharge de C_L à travers T_N

Si $V_{DD} = 1{,}8$ V et $V_{TN} = 0{,}5$ V

$$\boxed{\begin{aligned} t_f &\approx 3 R_{DS0N} C_L \\ t_{pHL} &\approx 1{,}5 R_{DS0N} C_L \approx \frac{t_f}{2} \end{aligned}}$$

N. B. Dans un circuit intégré numérique CMOS, la longueur de canal est la même pour tous les transistors, NMOS ou PMOS. D'autre part, $V_{TP} \approx -V_{TN}$, d'où :

$$\boxed{\frac{t_r}{t_f} = \frac{\mu_{Ns}}{\mu_{Ps}} \frac{W_N}{W_P} \approx 3 \frac{W_N}{W_P}}$$

Dans ces conditions, l'inverseur est électriquement symétrique si $W_P = 3W_N$. Les temps de commutation en montée et en descente sont alors égaux : $\boxed{t_f = t_r}$ et $\boxed{t_{pHL} = t_{pLH}}$.

A. N. Pour $\beta_P = \beta_N = 300\ \mu$A / V^2 et $C_L = 50$ fF, $t_r = t_f \approx 0{,}11$ ns, $t_{pLH} = t_{pHL} \approx 5 \times 10^{-2}$ ns.

En logique CMOS, les temps de commutation en montée et en descente sont du même ordre de grandeur (voire égaux si l'opérateur est électriquement symétrique), ce qui n'est pas le cas pour les logiques NMOS ou PMOS à charge résistive. Le temps de montée est bien inférieur au temps de montée de l'inverseur NMOS, et le temps de descente est bien inférieur au temps de descente de l'inverseur PMOS. L'association des 2 transistors complémentaires permet de construire une porte logique plus rapide.

3.5 Construction de fonctions combinatoires en logique CMOS

3.5.1 Notation

Pour construire les opérateurs CMOS, on associe toujours le même nombre de transistors NMOS et PMOS. Pour distinguer aisément les deux types de transistors, on adopte en général le symbolisme simplifié de la figure 3.33.

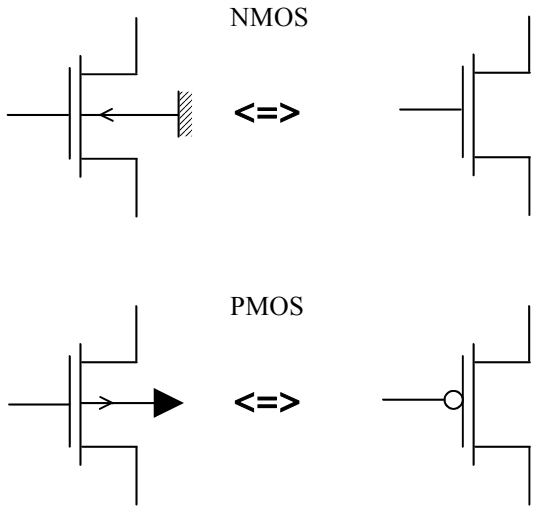


figure 3.33 : notation simplifiée des transistors MOS

3.5.2 Opérateurs statiques

La conservation des caractéristiques principales de l'inverseur CMOS, notamment pas de consommation au repos, impose pour la réalisation des opérateurs la constitution d'un réseau de NMOS et d'un réseau de PMOS tels que :

- Lorsque le réseau N est passant, le réseau P doit être bloqué afin qu'aucun courant ne circule dans le circuit et que $S = V_{SS} = 0$ V,
- Lorsque le réseau P est passant, le réseau N doit être bloqué afin qu'aucun courant ne circule dans le circuit et que $S = V_{DD}$.

Pour ce faire, chaque entrée de la porte logique est reliée à la grille de 2 transistors de type opposé, et les deux réseaux N et P ont des structures duales.

Un transistor NMOS « transmet » un 0 logique vers la sortie d'une porte logique lorsqu'un 1 logique est appliqué sur sa grille et un transistor PMOS « transmet » un 1 logique vers la sortie lorsqu'un 0 logique est appliqué sur sa grille. En conséquence, un circuit MOS réalise toujours une fonction logique complémentée.

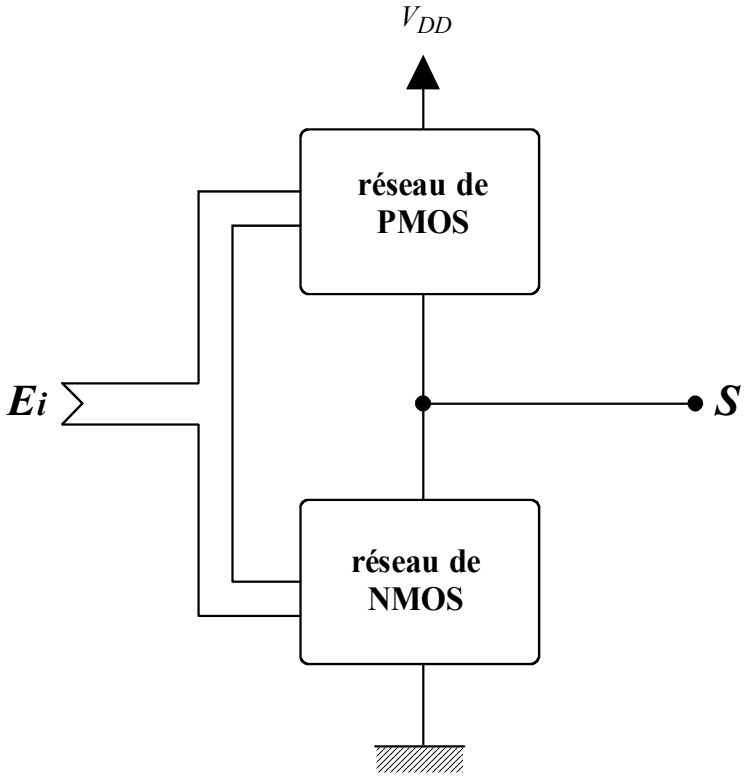


figure 3.34 : structure générale d'une porte logique CMOS

3.5.2.1 Porte NON ET ou NAND

La figure 3.35 donne la structure d'une porte NAND CMOS à 2 entrées. Son fonctionnement logique est le suivant :

- Lorsque les entrées *A* et *B* sont au niveau logique 1, les 2 transistors N sont passants, les 2 transistors P sont bloqués, $S = V_{SS}$ (niveau logique 0),
- Lorsque *A* vaut 0 ou *B* vaut 0, l'un au moins des transistors P est passant et l'un au moins des transistors N est bloqué, le réseau P est donc globalement passant et le réseau N globalement bloqué, $S = V_{DD}$ (niveau logique 1).

La variable logique *S* s'exprime donc comme $S = \overline{A} + \overline{B} = \overline{AB}$.

De manière similaire, une porte NAND à 3 entrées est constituée de 3 transistors N placés en série et de 3 transistors P placés en parallèle.

Pour réaliser la fonction ET, il faut associer une porte NAND et un inverseur. Deux **couches logiques** sont donc nécessaires pour construire une porte ET.

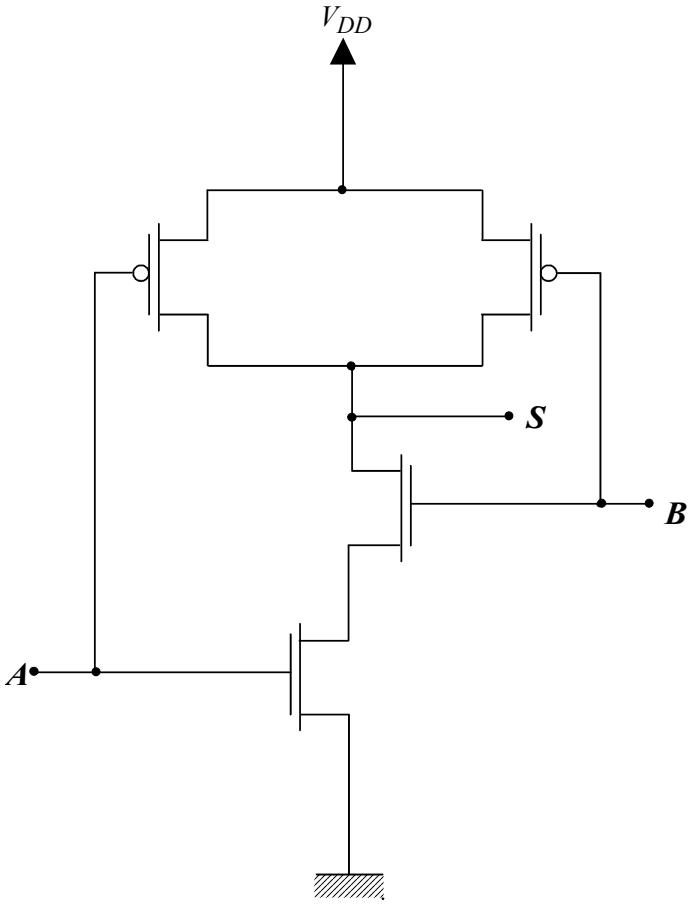


figure 3.35 : porte NAND CMOS à 2 entrées

3.5.2.2 Porte NON OU ou NOR

La figure 3.36 donne la structure d'une porte NOR CMOS à 2 entrées. Son fonctionnement logique est dual de celui de la porte NAND :

- Lorsque les entrées A et B sont au niveau logique 0, les 2 transistors P sont passants, les 2 transistors N sont bloqués, $S = V_{DD}$ (niveau logique 1),
- Lorsque A vaut 1 ou B vaut 1, l'un au moins des transistors N est passant et l'un au moins des transistors P est bloqué, le réseau N est donc globalement passant et le réseau P globalement bloqué, $S = V_{SS}$ (niveau logique 0).

On en déduit $S = \overline{A}.\overline{B} = \overline{A + B}$.

De manière similaire, une porte NOR à 3 entrées est constituée de 3 transistors N placés en parallèle et de 3 transistors P placés en série.

La fonction OU est obtenue en faisant suivre la porte NOR d'un inverseur.

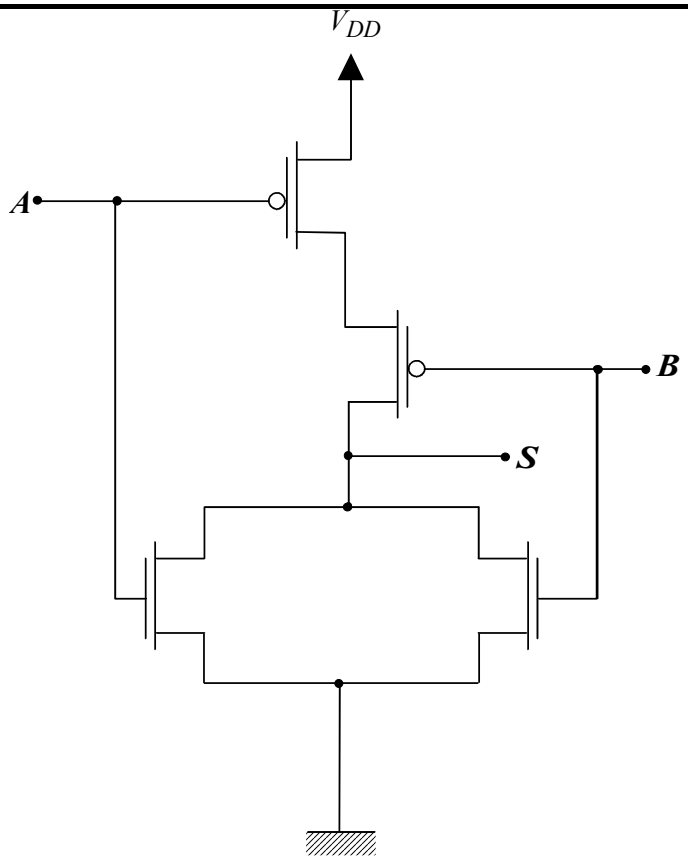


figure 3.36 : porte NOR CMOS à 2 entrées

3.5.2.3 Autres fonctions combinatoires

Pour construire en logique CMOS des fonctions combinatoires complexes, deux principales approches sont possibles, suivant les outils de conception utilisés :

- Le concepteur dispose d'une bibliothèque de portes logiques élémentaires, dont il assemble les éléments pour construire les fonctions combinatoires,
- Le concepteur a accès au niveau transistor et a la possibilité de construire directement des fonctions logiques plus complexes que les portes de base NAND, NOR et inverseur.

La construction directe des fonctions combinatoires permet de réaliser des blocs logiques moins encombrants et plus rapides, car nécessitant moins de couches logiques, que par assemblage de portes élémentaires. En revanche, la seconde approche ne requiert pas une connaissance approfondie de la réalisation des portes à partir des transistors.

Dans le cadre de la première approche, la méthode à appliquer pour construire $S = \mathrm{f}(E_i)$ est la suivante :

- Si la fonction f peut s'écrire sous la forme d'un complément, la synthèse de f est directe, sinon synthétiser $\bar{\mathrm{f}}$ et faire suivre d'un inverseur.
- Placer les transistors NMOS :
  - en série pour réaliser les ET des entrées,

- en parallèle pour réaliser les OU.

- Placer les transistors PMOS selon une structure duale :
  - en parallèle pour réaliser les ET des entrées,
  - en série pour réaliser les OU.

Le circuit doit comporter autant de transistors NMOS que PMOS.

- **Exemple 1** : synthèse de $S = AB + C$

S ne s'écrit pas sous la forme d'un complément, on synthétise donc $\overline{S}$ et on inverse. La réalisation de S requiert donc 2 couches logiques.

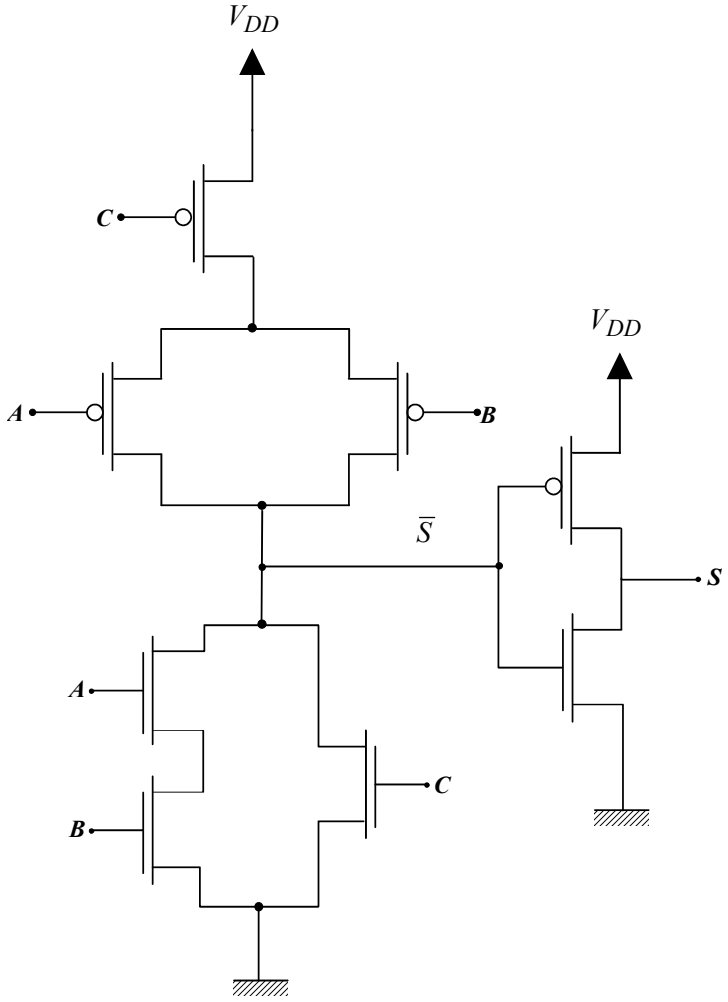


figure 3.37 : résultat de la synthèse de $S = AB + C$

- **Exemple 2** : synthèse de $S = \overline{A}\ \overline{B} + \overline{C}\ \overline{D}$

S peut s'écrire sous la forme d'un complément : $S = \overline{(A + B)(C + D)}$, la synthèse est donc directe (une seule couche logique).

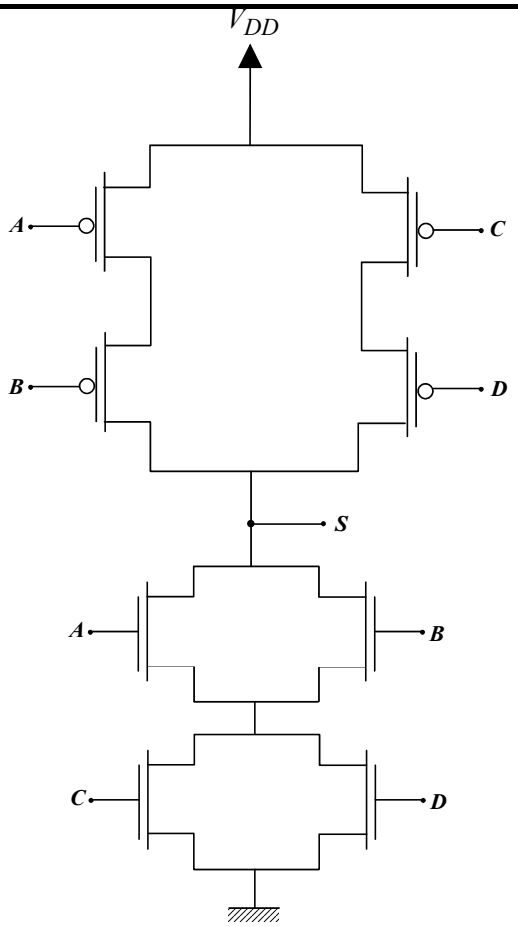


figure 3.38 : résultat de la synthèse de $S = \overline{A}\ \overline{B} + \overline{C}\ \overline{D}$

3.5.3 Opérateurs à base d'interrupteurs

L'utilisation de **portes de transfert** ou **interrupteurs** CMOS dans la construction de fonctions combinatoires peut dans certains cas simplifier considérablement leur structure et améliorer leurs performances dynamiques. Les interrupteurs sont également très utilisés pour la réalisation d'opérateurs à « sortie 3 états », et des circuits séquentiels élémentaires (bascules, cf. chapitre 5).

3.5.3.1 L'interrupteur CMOS ou porte de transfert

La structure de la porte de transfert CMOS est donnée en figure 3.39 (a).

D'un point de vue logique la porte de transfert présente deux modes de fonctionnement normal :

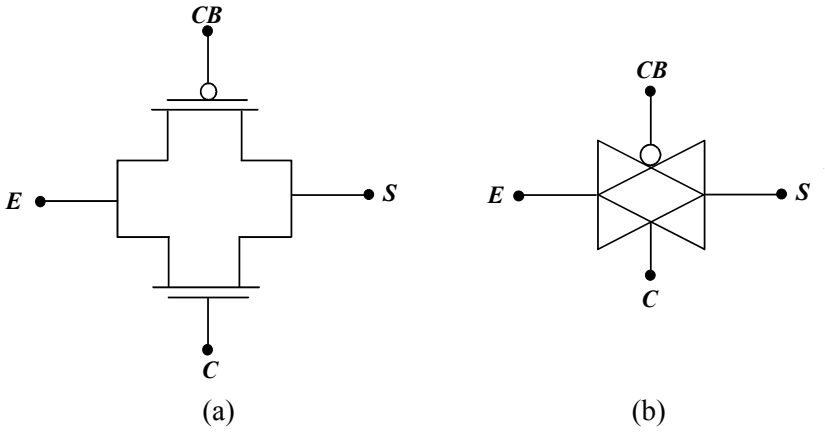
- Si $C = 0$ et $CB = 1$, l'interrupteur est ouvert,
- Si $C = 1$ et $CB = 0$, l'interrupteur est fermé.

Dans le cas où $CB \neq \overline{C}$, l'un des transistors est passant et l'autre bloqué, la porte de transfert est globalement passante, mais son fonctionnement est dégradé d'un point de vue électrique :

- Lorsque $C = CB = 1$ (tension V_{DD}), seul le transistor NMOS peut être passant. Il est en régime ohmique si un 0 logique (tension $V_{SS} = 0\,\text{V}$) est appliqué en entrée de l'interrupteur et alors $S = E$. Mais dans le cas où la tension V_{DD} est appliquée en entrée, le transistor est en régime saturé (à la limite du blocage) et la tension maximale en sortie vaut $V_{DD} - V_{TN}$, le niveau de sortie est alors dégradé.

- Lorsque $C = CB = 0$ (tension V_{SS}), seul le transistor PMOS peut être passant. Il est en régime ohmique lorsqu'un 1 logique (tension V_{DD}) est appliqué en entrée de l'interrupteur, mais bloqué si on applique V_{SS} en entrée. Il faut appliquer une tension au moins égale à $-V_{TP}$ pour que l'interrupteur devienne passant et que $S = E$.

L'association des deux types de transistors, avec des commandes complémentaires, permet de masquer le défaut de chacun d'eux. Les signaux appliqués à l'entrée de l'interrupteur CMOS sont restitués en sortie sans dégradation.



(a) (b)

figure 3.39 : porte de transfert ou interrupteur CMOS
(a) schéma électrique
(b) symbole

3.5.3.2 Fonction OU exclusif

L'interrupteur CMOS permet la réalisation d'opérateurs OU exclusifs et de multiplexeurs (cf. chapitre 4, section 3.1) beaucoup moins encombrants que par l'application de la méthode précédemment décrite. La figure 3.40 montre la réalisation d'un OU exclusif à l'aide de 8 transistors, 4 NMOS et 4 PMOS, contre 12 par la méthode classique.

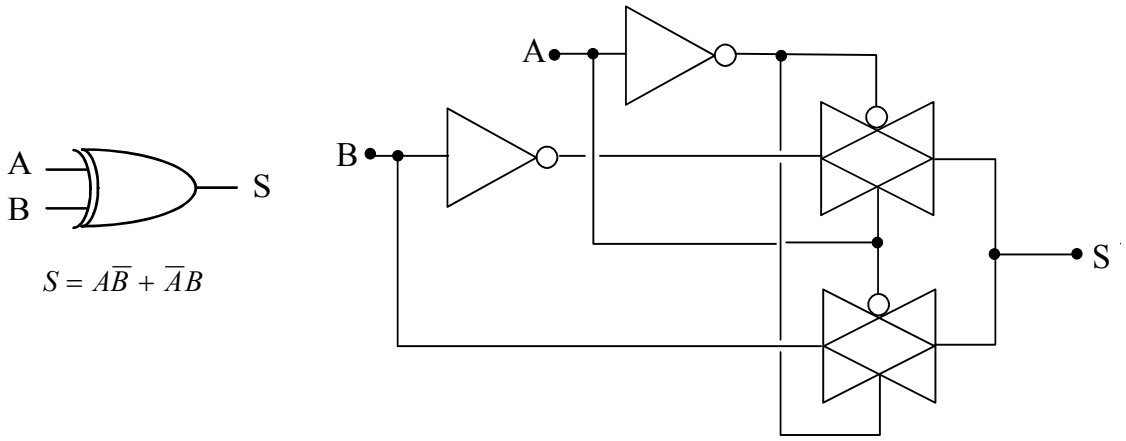


figure 3.40 : réalisation d'un OU exclusif à l'aide d'interrupteurs CMOS

3.5.3.3 Opérateurs à sortie 3 états

L'interrupteur CMOS est couramment utilisé pour réaliser des opérateurs dont on peut déconnecter la sortie, dits **opérateurs à sortie 3 états**. La figure 3.41 présente la structure d'un inverseur à sortie 3 états. Lorsque $T = 1$, l'interrupteur est fermé et $S = S'$. Lorsque $T = 0$, l'interrupteur est ouvert et la sortie S est déconnectée de S', on parle alors d'**état haute impédance** (notation : $S = Z$). Ce type d'opérateur permet d'interconnecter plusieurs blocs logiques sur une même ligne pour la réalisation de **bus**. Un seul opérateur à la fois doit être connecté au bus pour éviter les conflits.

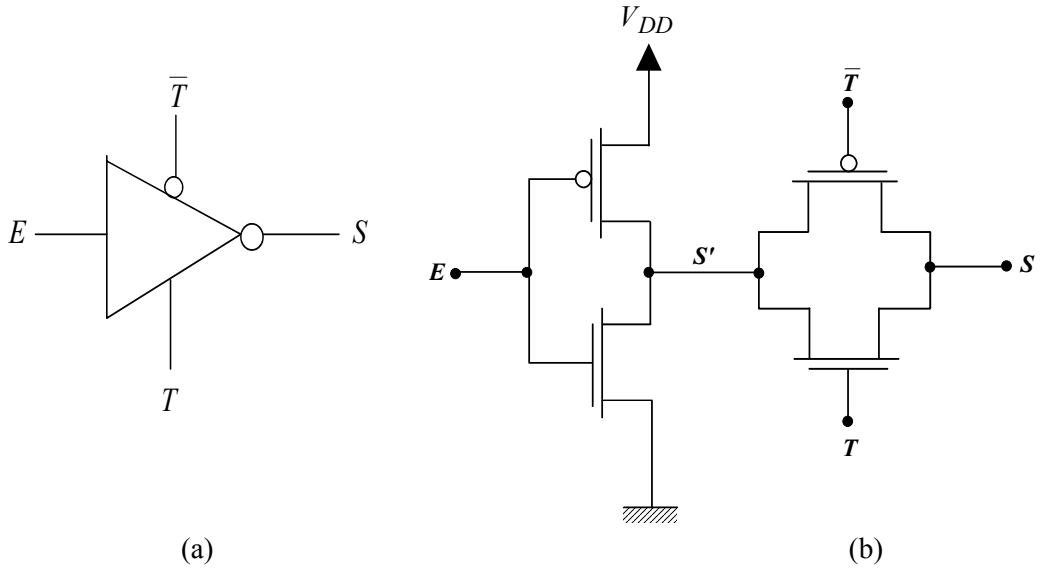


figure 3.41 : inverseur CMOS 3 états
(a) symbole (b) structure

3.6 Performances statiques et dynamiques des circuits logiques CMOS

Nous nous proposons de donner quelques éléments permettant d'estimer les principales performances statiques et dynamiques des circuits logiques CMOS, à partir des résultats de l'étude de l'inverseur.

3.6.1 Performances statiques

Les résultats obtenus pour l'inverseur CMOS sont en grande partie transposables aux autres portes logiques et à l'ensemble des circuits CMOS. Notamment :

- Dans un circuit CMOS, les niveaux logiques 1 et 0 sont représentés par les tensions V_{DD} et V_{SS},
- Les circuits CMOS ne consomment pas en statique.

3.6.2 Performances dynamiques

3.6.2.1 Puissance dynamique

L'expression de la puissance consommée en dynamique par une porte logique CMOS constituée d'une seule couche logique est la même que pour l'inverseur, à savoir :

$$\boxed{P_{dyn} = f_S V_{DD}^2 C_L}$$

où f_S représente la fréquence de commutation de la **sortie** de la porte. Dans le cas de l'inverseur, f_S représente également la fréquence de commutation de l'entrée de la porte, ce n'est pas le cas pour une porte logique quelconque, car une commutation en entrée n'entraîne pas nécessairement une commutation en sortie.

Dans le cas plus général d'un circuit piloté par un signal de référence (**horloge**) de fréquence *f*, si α représente le nombre moyen de transitions montantes (ou descendantes) de la sortie de la porte considérée par période de l'horloge, la puissance consommée par cette porte est égale à :

$$\boxed{P_{dyn} = \alpha f V_{DD}^2 C_L}$$

α est le **taux d'activité** de la porte logique.

La puissance totale consommée par un circuit est obtenue en sommant les puissances élémentaires consommées par chaque porte logique.

3.6.2.2 Temps de montée et de descente

Le calcul complet de t_f et t_r a été détaillé dans le cas de l'inverseur, mais il devient vite très lourd pour des portes plus complexes. En pratique, il n'est pas nécessaire de refaire ce calcul dans

chaque cas. On obtient en effet une bonne approximation en considérant que, lors des commutations, chaque transistor passant se comporte comme une résistance.

Dans le cas de l'inverseur, on a montré que

- Le temps de descente est le même que celui d'un réseau *RC* pour une résistance $R_N \approx 1{,}3R_{DS0N}$,
- Le temps de montée est le même que celui d'un réseau *RC* pour une résistance $R_P \approx 1{,}3R_{DS0P}$.

Les valeurs des temps de commutation pour une porte quelconque sont obtenues en remplaçant chaque transistor passant par son modèle de résistance équivalente et en calculant la résistance équivalente totale de l'étage passant. On notera R_f la résistance équivalente de l'étage N pour la descente, et R_r la résistance équivalente de l'étage P pour la montée. Avec ces notations :

$$\boxed{\begin{aligned} t_f &\approx \ln 9 R_f C_L \\ t_r &\approx \ln 9 R_r C_L \end{aligned}}$$

Par exemple, dans le cas de la porte NAND à 2 entrées (figure 3.35) :

- Pour la transition descendante, les deux transistors NMOS sont passants. La résistance équivalente R_f de l'étage N est alors égale à $2R_N$, si les 2 transistors NMOS sont identiques. En comparaison avec un inverseur réalisé avec les mêmes transistors : $t_f(NAND2) = 2t_f(INV)$.
- Pour la transition montante, deux configurations sont possibles :
  1. les deux transistors PMOS sont passants, la résistance équivalente de l'étage P vaut $R_r = \frac{R_P}{2}$, et $t_r(NAND2) = \frac{t_r(INV)}{2}$,
  2. un seul transistor PMOS est passant, la résistance équivalente de l'étage P vaut $R_r = R_P$, et $t_r(NAND2) = t_r(INV)$.

On obtient des résultats inverses dans le cas d'une porte NOR.

Dans le cas d'un bloc logique complexe, les temps de commutation sont donnés par la dernière couche logique du bloc.

3.6.2.3 Temps de propagation. Notions d'entrance et de sortance

Lors de la conception d'un bloc logique, un paramètre fondamental pour le concepteur est son temps de propagation, c'est-à-dire le temps nécessaire pour qu'un changement des variables logiques en entrée soit répercuté en sortie. Ce temps est représentatif de la vitesse maximale de fonctionnement du bloc. Cette section est consacrée à la présentation des méthodes de calcul appliquées en pratique par les concepteurs de systèmes numériques lorsqu'ils utilisent des bibliothèques de portes.

3.6.2.4 Analyse de la capacité de charge C_L d'une porte logique.

- **Capacité de sortie d'une porte logique**

On appelle **capacité de sortie** C_S d'une porte logique la capacité vue en régime dynamique par cette porte logique sur son nœud de sortie lorsqu'elle n'attaque aucune autre porte. C_S est égale à la somme des capacités parasites des transistors de la porte reliés à la sortie, et est essentiellement constituée des **capacités de jonctions** de ces transistors. Par exemple, dans le cas d'un inverseur CMOS (figure 3.42), la capacité de sortie est constituée des capacités de jonction drain/substrat des deux transistors N et P (cf. section 2.3.2). Les capacités de jonctions source/substrat n'interviennent pas ici car elles sont court-circuitées.

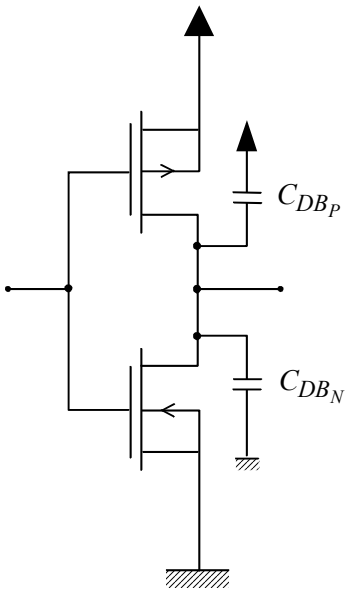


figure 3.42 : capacité de sortie d'un inverseur CMOS

Pour une porte logique quelconque,

$$\boxed{C_S = \sum C_{j_N} + \sum C_{j_P}}$$

où C_{j_N} et C_{j_P} représentent les capacités de jonction des transistors NMOS et PMOS reliés à la sortie. Ces capacités sont **proportionnelles à la surface des diffusions** et donc à la largeur des transistors (cf. 2.3.2).

- **Capacité d'entrée d'une porte logique**

On appelle **capacité d'entrée** C_E d'une porte logique, relativement à une de ses entrées, la capacité vue en régime dynamique sur les grilles des transistors reliés à cette entrée. Dans le cas de l'inverseur CMOS, les différentes capacités intervenant dans la capacité d'entrée sont détaillées sur la figure 3.43.

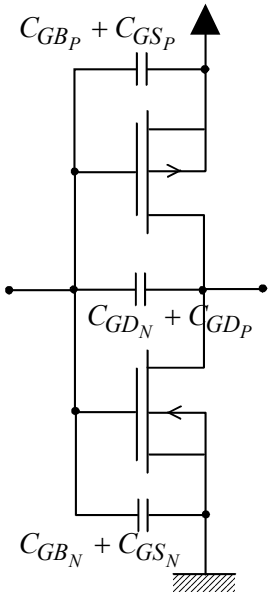


figure 3.43 : capacités intervenant dans la capacité d'entrée d'un inverseur CMOS

D'après la section 2.3.1 :

$$C_E = C_{G_N} + C_{G_P}$$
$$C_E \approx W_N L_N C_{ox} + W_P L_P C_{ox}$$

où W_N, W_P, L_N, et L_P désignent les largeurs et longueurs de canal des transistors NMOS et PMOS de l'inverseur.

Pour une porte logique quelconque, la capacité d'entrée C_E s'écrit sous la forme :

$$\boxed{C_E = \sum C_{G_N} + \sum C_{G_P}}$$

où C_{G_N} et C_{G_P} représentent les capacités de grille des transistors NMOS et PMOS connectés à l'entrée considérée.

- **<u>Capacité totale de charge d'une porte logique</u>**

La capacité totale de charge C_L d'une porte logique dans un circuit numérique peut être exprimée comme la somme de trois termes :

- La capacité de sortie de la porte,
- La capacité parasite d'interconnexion de la porte avec les portes en charge,
- La somme des capacités d'entrée des portes en charge.

A titre d'exemple, dans le cas de la figure 3.44, la capacité de charge C_L de la porte logique ❶ est égale à $C_L = C_{S1} + C_{int} + C_{E2} + C_{E3} + C_{E4}$, où :

- C_{S1} est la capacité de sortie de ❶,

- C_{int} est la capacité d'interconnexion, proportionnelle à la longueur des pistes de connexion entre ❶ et ❷, ❸, et ❹,
- C_{E2}, C_{E3}, et C_{E4} sont les capacités d'entrée des portes ❷, ❸, et ❹.

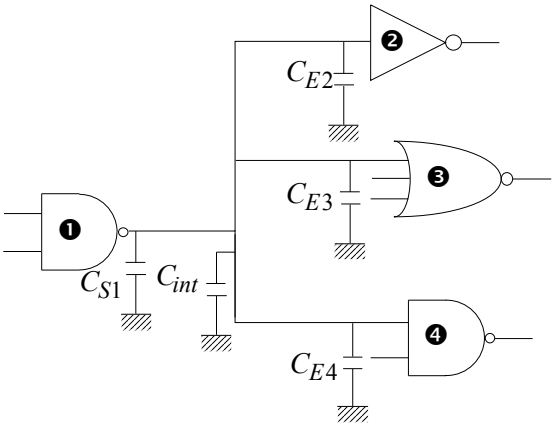


figure 3.44 : analyse de la capacité de charge d'une porte logique

3.6.2.5 Capacité d'entrée minimale, entrance, et sortance

Soit C_{min} la capacité d'entrée d'un opérateur (inverseur) de taille minimale, pour un procédé de fabrication CMOS donné. Dans un circuit intégré numérique, la longueur de canal des transistors N ou P est toujours minimale. Un opérateur de taille minimale est donc un opérateur dont les transistors sont de largeur minimale, et

$$C_{min} \approx L_{min} C_{ox} (W_{Nmin} + W_{Pmin})$$

On appelle **entrance** ou **fan-in** d'un opérateur logique, relativement à une de ses entrées, le rapport de la capacité C_E présentée à cette entrée à C_{min}.

$$\boxed{F_{in} = \frac{C_E}{C_{min}}}$$

A titre d'exemple, une porte NAND ou NOR constituée de transistors de taille minimale présente une entrance de 1 sur chacune de ses entrées : chaque entrée est reliée à 2 transistors, 1 NMOS et 1 PMOS, la capacité d'entrée est donc égale à C_{min}. Un inverseur de **taille double** ($W_N = 2W_{Nmin}$, $W_P = 2W_{Pmin}$) a une entrance de 2.

On appelle **sortance** ou **fan-out** d'un opérateur logique la somme des entrances des opérateurs en charge. Ainsi, dans l'exemple de la figure 3.44, si tous les transistors sont de taille minimale, les portes ❷, ❸, et ❹ ont chacune une entrance égale à 1 et la sortance de ❶ vaut 3.

3.6.2.6 Temps de propagation d'un bloc logique et sortance

La capacité totale de charge C_L d'une porte logique peut s'écrire sous la forme :

$$C_L = C_{int} + C_S + \sum C_E = C_{int} + C_S + C_{min} \sum F_{in} = C_{int} + C_S + F_{out} C_{min}$$

où F_{out} est le fan-out de l'opérateur.

Pour une porte logique élémentaire constituée d'une seule couche logique, les temps de propagation sont proportionnels à $R_f C_L$ pour t_{pHL}, à $R_r C_L$ pour t_{pLH}. En tenant compte de l'expression précédente de C_L, les différents temps de propagation peuvent être écrits sous la forme générale :

$$\boxed{t_p = t_{p\,\mathrm{int}} + t_{p0} + \tau F_{out}}$$

où $t_{p\,\mathrm{int}}$ est le retard lié aux connexions en sortie de la porte, t_{p0} est le retard intrinsèque lié aux capacités de jonctions des transistors en sortie de la porte, et τ représente le retard engendré par une charge minimale en sortie de la porte.

La valeur de $t_{p\,\mathrm{int}}$ n'est pas calculable à partir du seul schéma logique du circuit. Elle ne peut être calculée de façon précise qu'après l'étape de placement-routage du circuit, lorsque l'on dispose de la description topologique du circuit et que les longueurs des interconnexions entre portes sont connues.

La connaissance des paramètres τ et t_{p0} pour une porte de taille minimale permet de calculer les temps de propagation des portes du même type de toute taille, et dans toutes les conditions de charge. En effet, pour un opérateur de taille *k* (transistors *k* fois plus larges que le transistor de taille minimale) :

- le paramètre τ est multiplié par $\frac{1}{k}$ par rapport à l'opérateur de taille 1, car R_f et R_r sont inversement proportionnels à la largeur *W* des transistors,
- le paramètre t_{p0} est indépendant de *k* car les capacités de jonction des transistors sont proportionnelles à la largeur des diffusions et les termes $R_f C_S$ et $R_r C_S$ sont donc indépendants de *k*.

Ces deux paramètres sont fournis par les bibliothèques logiques des fondeurs, et sont utilisés par les outils de conception assistée par ordinateur (CAO) pour calculer les temps de propagation dans les portes logiques d'un circuit.

Exemple Soient t_{p0} et τ les paramètres correspondant au temps de propagation moyen d'un inverseur de taille minimale. On supposera que, dans le cas traité, les retards liés aux interconnexions entre les portes logiques sont négligeables. Déterminer le temps de propagation moyen du montage de la figure 3.45 en fonction de sa sortance F_{out}.

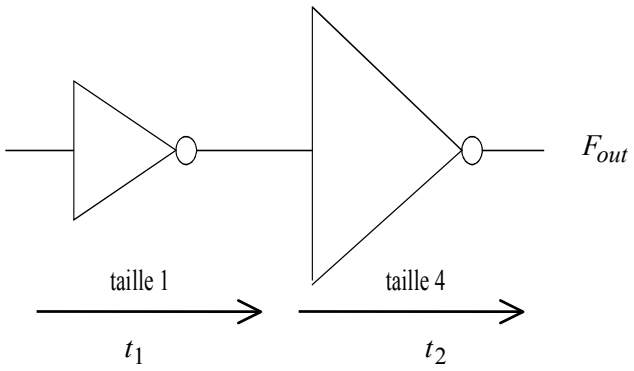


figure 3.45 : exemple de calcul de temps de propagation

Temps de propagation du premier inverseur : $t_1 = t_{p0} + 4\tau$

Temps de propagation du second inverseur : $t_2 = t_{p0} + \frac{\tau}{4} F_{out}$

Temps de propagation total : $t_p = \underbrace{2t_{p0} + 4\tau}_{t'_{p0}} + \underbrace{\frac{\tau}{4}}_{\tau'} F_{out}$

Cet exemple permet de mettre en évidence que l'écriture des temps de propagation sous la forme $t_p = t_{p0} + \tau F_{out}$ est en fait applicable pour tout bloc logique, quelle que soit sa complexité.

3.7 Evolution des performances des circuits CMOS

Les progrès technologiques permettent de réduire les dimensions des motifs lithographiques des circuits dans un rapport $k \approx 1{,}5$ tous les 3 ans environ. Les dimensions latérales (L et W : longueur et largeur de canal des transistors) aussi bien que verticales (T_{ox} : épaisseur de l'oxyde de grille) sont affectées par cette réduction (cf. tableau 3.1, page 49). Ces évolutions ont une répercussion sur la densité d'intégration des circuits, ainsi que sur leur vitesse et leur consommation :

- Si les dimensions W et L des transistors sont divisées par k, leur surface est divisée par k^2, et la densité d'intégration est donc multipliée par k^2.
- La vitesse des portes logiques est inversement proportionnelle au temps de propagation. Or, $t_p \propto R_{DS0} C_L$, et $R_{DS0} = \frac{1}{\beta(V_{DD} - |V_T|)} = \frac{L}{\mu C_{ox} W (V_{DD} - |V_T|)}$. Le terme prépondérant dans l'expression de C_L est dû à la capacité d'entrée des portes en charge, soit proportionnel à WLC_{ox}. Si la tension d'alimentation est inchangée, la vitesse d'une porte varie donc comme $\frac{1}{L^2}$. Elle est multipliée par k^2. En fait, pour des dimensions submicroniques, lorsque la longueur de canal des transistors diminue, la mobilité effective des porteurs dans le canal diminue également, et en pratique la vitesse des portes logiques

est accrue dans un rapport k^i, où i est compris entre 1 et 2, tendant vers 1 lorsque L décroît.

- Si on définit la **puissance de calcul** d'un circuit comme le produit de sa complexité (nombre d'opérateurs) par sa vitesse, celle-ci est alors accrue dans un rapport k^4, tendant vers k^3.
- La consommation d'une porte logique est proportionnelle à C_L, soit à WLC_{ox}. C_{ox} est inversement proportionnel à l'épaisseur de l'oxyde de grille T_{ox}. La puissance consommée diminue donc dans un rapport k pour une fréquence de fonctionnement et une tension d'alimentation inchangées.

La réduction des géométries sur un circuit intégré permet donc *a priori* de gagner sur tous les tableaux : surface, consommation, et vitesse. Néanmoins, un problème crucial se pose du point de vue de la consommation : si l'on profite de l'accroissement de la vitesse des circuits pour augmenter d'autant leur fréquence de fonctionnement, et sachant que la complexité des circuits est également accrue à surface constante, la consommation totale des circuits intégrés s'accroît en fait dans un rapport de k^3, tendant vers k^2. La consommation est donc actuellement le facteur principal qui limite la croissance de la densité d'intégration des circuits.

Pour pallier ce problème, la solution adoptée en pratique consiste à diminuer la tension d'alimentation. La puissance consommée par un opérateur logique élémentaire étant proportionnelle à V_{DD}^2, la diminution de la valeur de V_{DD} permet de réduire sensiblement la consommation. En revanche, une tension d'alimentation plus faible implique une détérioration des performances dynamiques du circuit, mais avec une perte inférieure au gain en consommation, puisque la vitesse des opérateurs est proportionnelle à $V_{DD} - |V_T|$. La tension d'alimentation est passée de 5 V à 2,5 V pour les technologies correspondant à des longueurs de grille L comprises entre 0,5 µm et 0,25 µm, à 1,8 V pour L = 0,18 µm et à 1,2 V pour L = 0,13 µm. La diminution de V_{DD} doit également s'accompagner d'une réduction de la tension de seuil des transistors. Les prévisions d'évolution de la valeur de V_{DD} dans les quinze prochaines années sont présentées dans le tableau 3.1, page 49.

L'accroissement de la vitesse des opérateurs CMOS a également des conséquences sur la conception des circuits. En effet, pour les technologies en deçà de 0,5 µm, les temps de propagation dans les portes logiques sont devenus inférieurs aux retards liés aux interconnexions entre portes. Ceci signifie que les performances dynamiques des circuits dépendent essentiellement de l'étape de placement-routage du circuit, étape située en fin de la chaîne de conception, et non du schéma logique. La solution à ce problème doit en partie être apportée par les nouvelles générations d'outils de CAO qui ont à prendre en compte cette contrainte. D'autre part, certains fondeurs, IBM notamment, commencent à réaliser des circuits à interconnexions en cuivre, matériau moins résistif que l'aluminium, qui engendre des retards moindres.

4. Bibliographie

[DOC95] J.-P. Dauvin, J. Olliver, et D. Coulon, *Les composants électroniques et leur industrie*, Collection Que sais-je, PUF, Paris, 1995.

[Haz91] H. Hazdenar, *Digital Microelectronics*, The Benjamin Cummings publishing company, Inc., 1991.

[ITRS13] *The International Technology Roadmap for Semiconductors,* 2013 Edition. http://www.itrs.net/Links/2013ITRS/

[LXZ99] S. Luryi, J. Xu, and A. Zaslavsky, *Future Trends in Microelectronics – The Road Ahead*, Wiley-Interscience Pub, 1999.

[NR94] C. Nowakowski et A. Roux, *Histoire des systèmes de télécommunication*, Lavoisier, 1994.

[RP96] J. M. Rabaey and M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996.

Chapitre 4 : Fonctions et circuits combinatoires

1. Définitions

On appelle **opérateur** ou **fonction combinatoire** un opérateur logique dont les sorties ne dépendent, à chaque instant, que de l'état de ses entrées. A chaque configuration des entrées correspond une configuration des sorties et une seule. Un circuit numérique réalisant une fonction combinatoire est un **circuit combinatoire**.

Outre les opérateurs élémentaires cités au chapitre 2, on distingue comme opérateurs combinatoires standard :

- les opérateurs de transcodage,
- les opérateurs d'aiguillage,
- les opérateurs de comparaison,
- les opérateurs arithmétiques.

Avertissement : L'étude de ces différents opérateurs est illustrée par des exemples de circuits standard issus de catalogues de fabricants de circuits intégrés. Les références complètes de ces circuits contiennent, outre un numéro caractéristique de la fonctionnalité du circuit, des informations complémentaires inutiles dans le cadre de ce chapitre (fabricant, technologie de fabrication, produit commercial ou militaire, etc.). Nous nous limiterons donc, ici, à fournir une référence générique n'indiquant que la fonctionnalité du circuit. A titre d'exemple, « *XX* 85 » représente tous les circuits dont la référence se termine par 85, numéro désignant un comparateur de deux mots de 4 bits. Les fiches techniques (**data sheets**) des circuits cités dans ce chapitre peuvent être consultées dans des catalogues tels que [Mot], [TI], ou [NS], par exemple.

2. Les opérateurs de transcodage

2.1 Définition

On désigne par opérateur de **transcodage** un opérateur qui traduit une information dans un code donné (**codeur** ou **encodeur**) ou bien qui, au contraire, permet d'extraire une ou des informations à partir d'un code donné (**décodeur**) ou bien encore réalise la conversion d'un code vers un autre (**convertisseur de code** ou **transcodeur**).

2.2 Les codeurs

Le principe de fonctionnement d'un codeur est le suivant : lorsqu'une entrée est activée, les sorties affichent la valeur correspondant au numéro de l'entrée dans le code binaire choisi. Un codeur peut être vu comme un convertisseur du code décimal vers un code binaire.

Une seule entrée du codeur doit normalement être activée à la fois. Dans le cas où le code en sortie est le code binaire pur, le circuit correspondant possède N entrées et n sorties, avec $2^{n-1} < N \leq 2^n$.

2.2.1 Exemples de codeurs

- **<u>Exemple 1</u> : codeur décimal vers binaire (10 entrées vers 4 sorties)**

Ce codeur reçoit un chiffre décimal sur une des dix entrées et génère l'équivalent binaire sur les sorties $A0$ à $A3$. Une seule entrée doit être active à la fois.

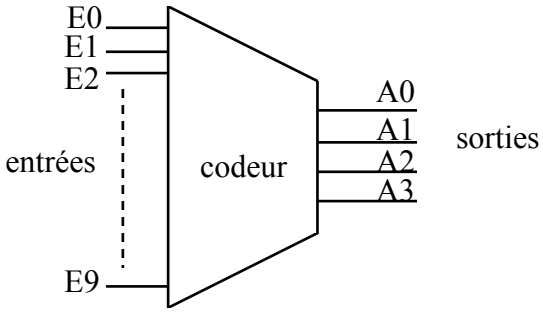


figure 4.1 : codeur décimal vers binaire

On trouvera ci-après (tableau 4.1) la table de vérité simplifiée et les équations de fonctionnement de ce type de codeur.

	Sorties			
Entrée activée (= 1)	$A3$	$A2$	$A1$	$A0$
$E0$	0	0	0	0
$E1$	0	0	0	1
$E2$	0	0	1	0
$E3$	0	0	1	1
$E4$	0	1	0	0
$E5$	0	1	0	1
$E6$	0	1	1	0
$E7$	0	1	1	1
$E8$	1	0	0	0
$E9$	1	0	0	1

tableau 4.1 : table de vérité réduite du codeur 10 vers 4

Equations logiques :

$$A0 = E1 + E3 + E5 + E7 + E9$$
$$A1 = E2 + E3 + E6 + E7$$
$$A2 = E4 + E5 + E6 + E7$$
$$A3 = E8 + E9$$

Cet opérateur, qui n'existe pas sous forme de circuit intégré standard, peut facilement être réalisé à partir de portes élémentaires.

- **<u>Exemple 2</u> : codeur binaire 8 vers 3**

Ce codeur reçoit une information codée sur une de ses huit entrées et génère l'équivalent binaire sur les sorties $A0$ à $A2$. Une seule entrée doit être active à la fois.

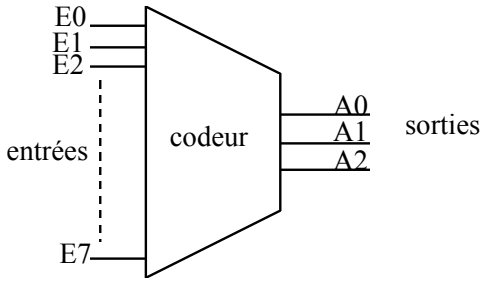


figure 4.2 : codeur 8 vers 3

	Sorties		
Entrée activée (= 1)	$A2$	$A1$	$A0$
$E0$	0	0	0
$E1$	0	0	1
$E2$	0	1	0
$E3$	0	1	1
$E4$	1	0	0
$E5$	1	0	1
$E6$	1	1	0
$E7$	1	1	1

tableau 4.2 : table de vérité réduite du codeur 8 vers 3

Equations logiques :

$$A0 = E1 + E3 + E5 + E7$$
$$A1 = E2 + E3 + E6 + E7$$
$$A2 = E4 + E5 + E6 + E7$$

Dans les deux exemples précédents il n'y a en théorie qu'une seule entrée active parmi N. Si, dans une application, plusieurs entrées peuvent être actives simultanément il faut utiliser un codeur **prioritaire**.

2.2.2 Codeur prioritaire

Ce type de codeur fixe un ordre de priorité entre les entrées. Dans le cas d'un encodage en binaire pur, le codeur prioritaire donne en général la priorité à l'entrée de poids le plus élevé. Par exemple, si les entrées 2, 8 et 9 sont activées simultanément, le codage de sortie correspondra à l'entrée 9. Ce circuit permet de détecter le rang du premier bit positionné à 1 dans un mot d'entrée.

- **Exemple : encodeur prioritaire 4 vers 2**

L'opérateur de la figure 4.3 est un encodeur prioritaire possédant 4 entrées ; l'entrée $E3$ correspond à l'entrée la plus prioritaire, et l'entrée $E0$ correspond à l'entrée la moins prioritaire

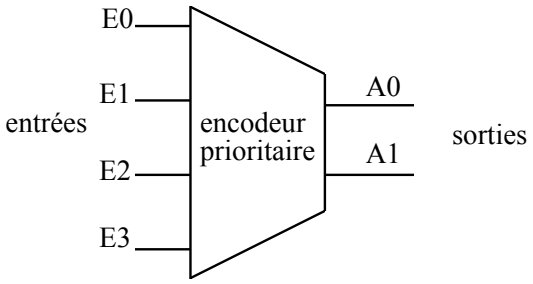


figure 4.3 : encodeur prioritaire 4 vers 2

Sa table de vérité et ses équations de fonctionnement sont présentées ci-après (tableau 4.3).

$E3$	$E2$	$E1$	$E0$	$A1$	$A0$
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	1	0	0

tableau 4.3: table de vérité de l'encodeur prioritaire 4 vers 2

Après simplification, nous obtenons les équations suivantes :

$$A0 = E3 + \overline{E2}\,E1$$
$$A1 = E3 + E2$$

2.3 Les décodeurs

Un décodeur est un opérateur à n entrées et N sorties avec $N \leq 2^n$. Une sortie du décodeur est activée lorsqu'une configuration particulière du code est affichée en entrée. Suivant le nombre de sorties, le décodeur peut décoder toutes les configurations possibles du code en entrée ou une partie seulement. Un décodeur à n entrées, permettant de décoder toutes les configurations du code binaire pur sur n bits, possède 2^n sorties. Une seule sortie est activée à la fois. En plus des n entrées, certains décodeurs possèdent une ou plusieurs entrées de validation. Par exemple, pour une validation active au niveau bas, si l'entrée de validation V vaut 0, alors le décodage est autorisé ; dans le cas contraire ($V = 1$), les sorties sont inhibées et restent inactives. Ces entrées de validation permettent de grouper plusieurs décodeurs afin d'augmenter l'amplitude de décodage (voir section 2.3.2).

2.3.1 Les principaux types de décodeurs

2.3.1.1 Les décodeurs binaires

Ce type d'opérateur permet d'associer une sortie parmi 2^n avec une information d'entrée codée sur *n* bits.

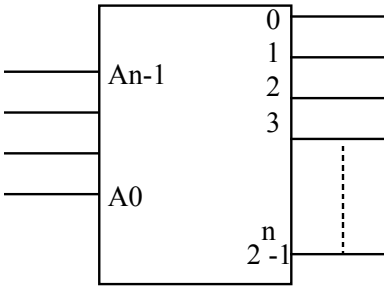


figure 4.4 : décodeur binaire « *n* vers 2^n » ou « 1 parmi 2^n »

- **Exemple 1 : décodeur 2 vers 4**

Le tableau 4.4 présente le fonctionnement d'un décodeur binaire 2 vers 4 ou 1 parmi 4.

$E1$	$E0$	$S0$	$S1$	$S2$	$S3$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

tableau 4.4 : table de vérité d'un décodeur binaire 2 vers 4

Dans cet exemple, on a choisi d'associer la valeur "0" lorsque la sortie est décodée (sorties actives à 0). Les équations de fonctionnement de ce circuit sont les suivantes :

$$\overline{S0} = \overline{E1}.\overline{E0}$$
$$\overline{S1} = \overline{E1}.E0$$
$$\overline{S2} = E1.\overline{E0}$$
$$\overline{S3} = E1.E0$$

La réalisation du logigramme de ce composant à l'aide d'opérateurs logiques élémentaires ne présente pas de difficulté particulière.

- **Exemple 2 : décodeur 3 vers 8**

Contrairement à l'exemple précédent, on associe la valeur logique "1" lorsque le rang de la sortie active correspond à la valeur binaire présentée en entrée (sorties actives à 1). La table de vérité (tableau 4.5) comporte 8 sorties qui correspondent aux mintermes des variables d'entrée.

Entrées			Sorties							
C	*B*	*A*	*S*0	*S*1	*S*2	*S*3	*S*4	*S*5	*S*6	*S*7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

tableau 4.5 : table de vérité du décodeur 3 vers 8

Les sorties du décodeur sont données par les relations suivantes :

$$S0 = \overline{C}\,\overline{B}\,\overline{A}$$
$$S1 = \overline{C}\,\overline{B}\,A$$
$$\ldots$$
$$S7 = C\,B\,A$$

Une réalisation possible du décodeur 3 vers 8 est immédiate :

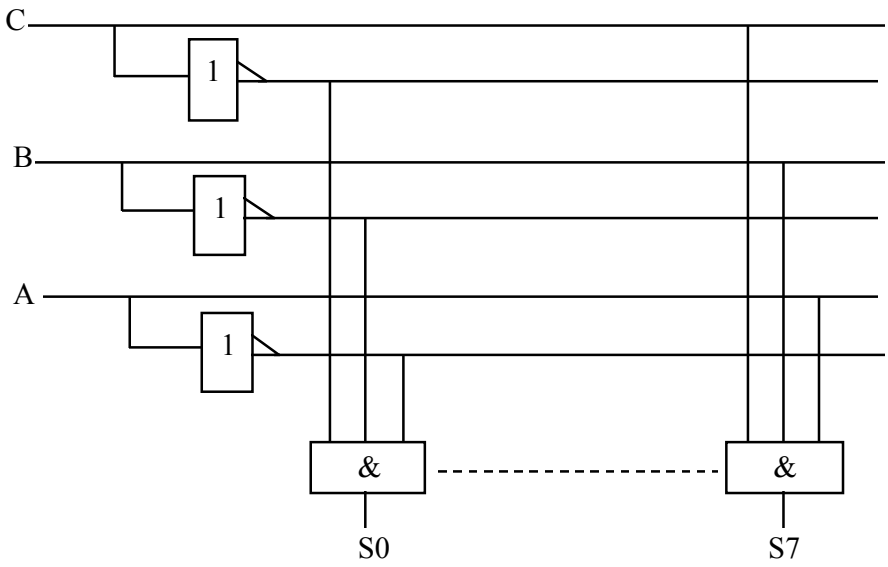


figure 4.5 : réalisation d'un décodeur 3 vers 8

2.3.1.2 Le décodeur BCD

Le code BCD (Binary-Coded Decimal, cf. chapitre 1, section 2.5.1.2) est utilisé pour coder les dix chiffres décimaux. Ce code à 4 bits laisse donc inutilisées 6 combinaisons sur les 16 possibles. Lorsqu'une combinaison, comprise entre 0 et 9, est appliquée sur les entrées *ABCD* (*A* est le bit de poids fort), la sortie correspondante est validée. Les sorties restent au repos (niveau 1 par exemple) dans le cas où une combinaison comprise entre 10 et 15 est appliquée sur les entrées.

Table de vérité :

A	B	C	D	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

tableau 4.6 : table de vérité du décodeur BCD

Représentation symbolique usuelle :

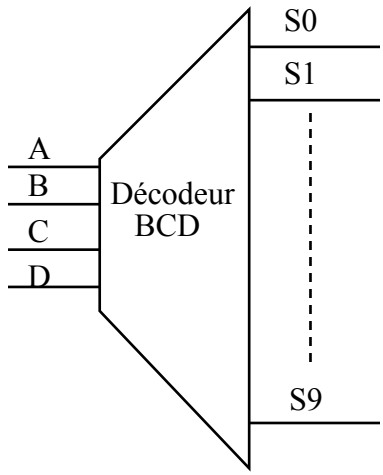


figure 4.6 : entrées et sorties du décodeur BCD

2.3.2 Matrice de décodage

Dès que l'on souhaite réaliser un décodeur de plus de 3 bits en entrée, il est conseillé d'utiliser une structure en matrice (figure 4.7) pour obtenir un opérateur de complexité matérielle minimale.

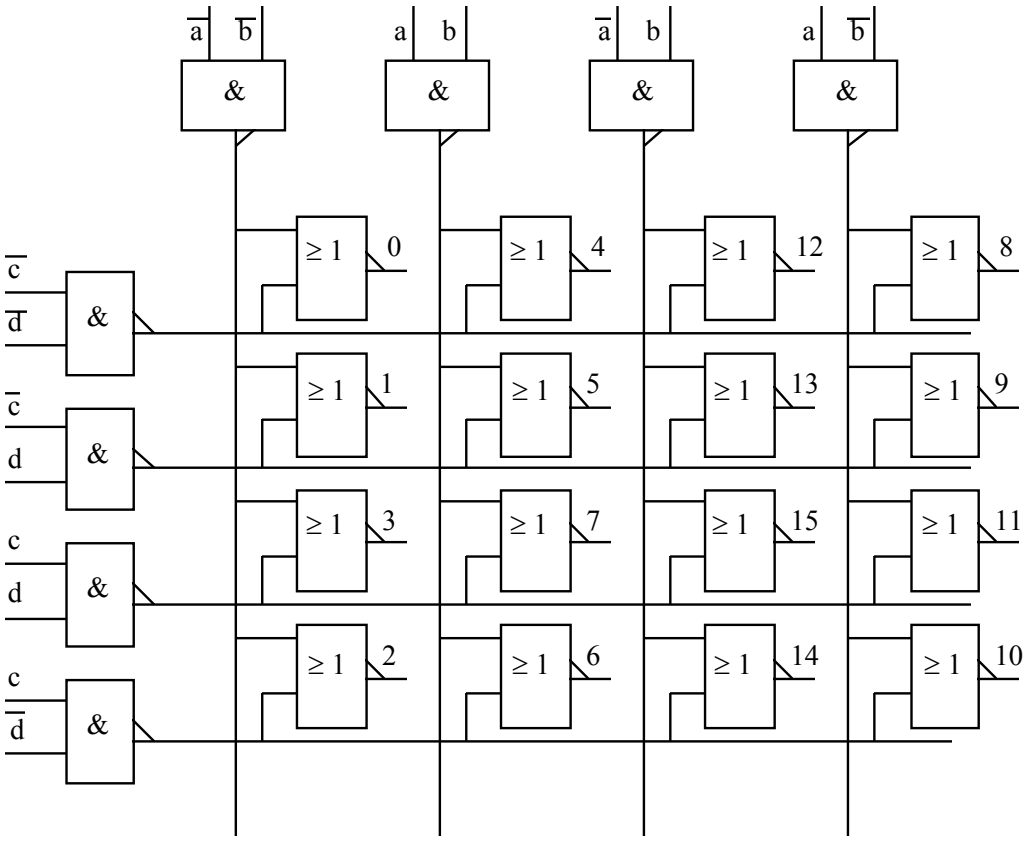


figure 4.7 : matrice de décodage de 4 bits

Dans la matrice de la figure 4.7, chaque case du tableau de Karnaugh ou chaque ligne de la table de vérité correspond à une structure matérielle composée de 2 portes NAND et d'une porte NOR. L'utilisation multiple des mintermes des variables *c* et *d*, ainsi que des variables *a* et *b*, permet réduire la complexité matérielle par rapport à un circuit décodant les seize sorties indépendamment les unes des autres. On notera également que tous les chemins entre entrées et sorties traversent des couches logiques identiques.

2.3.3 Association de décodeurs

Il est possible d'associer plusieurs décodeurs afin d'augmenter les capacités d'un système. La figure 4.8 montre comment deux décodeurs 1 parmi 16 (référence *XX* 154 des catalogues de circuits standard) sont associés pour former un décodeur de 1 parmi 32 (fonctionnalité qui n'existe pas chez les fabricants de circuits standard, mais qui peut être directement réalisée dans un circuit spécifique). Outre les entrées *A*, *B, C,* et *D*, le circuit *XX* 154 possède deux entrées de validation $\overline{G1}$ et $\overline{G2}$. La notation barrée ainsi que les ronds sur ces entrées signifient qu'elles sont actives au niveau logique 0 : les sorties du décodeur ne sont validées que lorsque $\overline{G1} = \overline{G2} = 0$.

Si l'entrée *a* vaut 0, les entrées $\overline{G1}$ et $\overline{G2}$ du décodeur **1** sont actives, celles du décodeur **2** inactives. Le décodeur **1** est validé, et le décodeur **2** inhibé. Les sorties $S0$ à $S15$ sont alors représentatives des entrées *b c d e*. Si l'entrée *a* vaut 1, seul le décodeur **2** est validé et les entrées *b*, *c*, *d*, et *e* sont décodées sur les sorties $S16$ à $S31$.

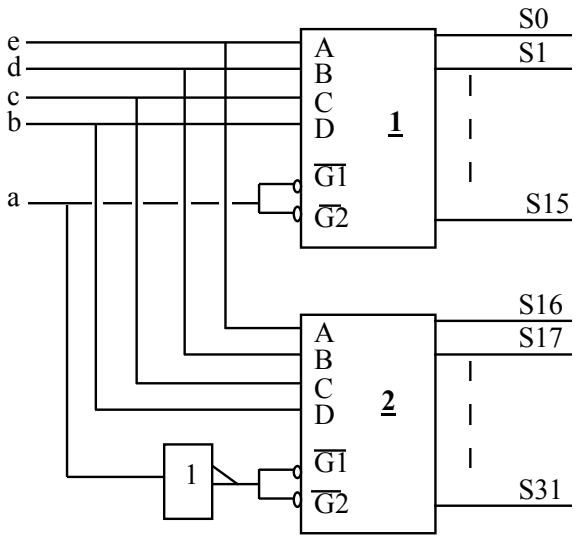


figure 4.8 : réalisation d'un décodeur 1 parmi 32 à partir de 2 décodeurs 1 parmi 16

2.3.4 Décodage à plusieurs niveaux

Afin de réaliser des circuits décodeurs de grande capacité, il est possible d'associer des décodeurs de base sur plusieurs niveaux.

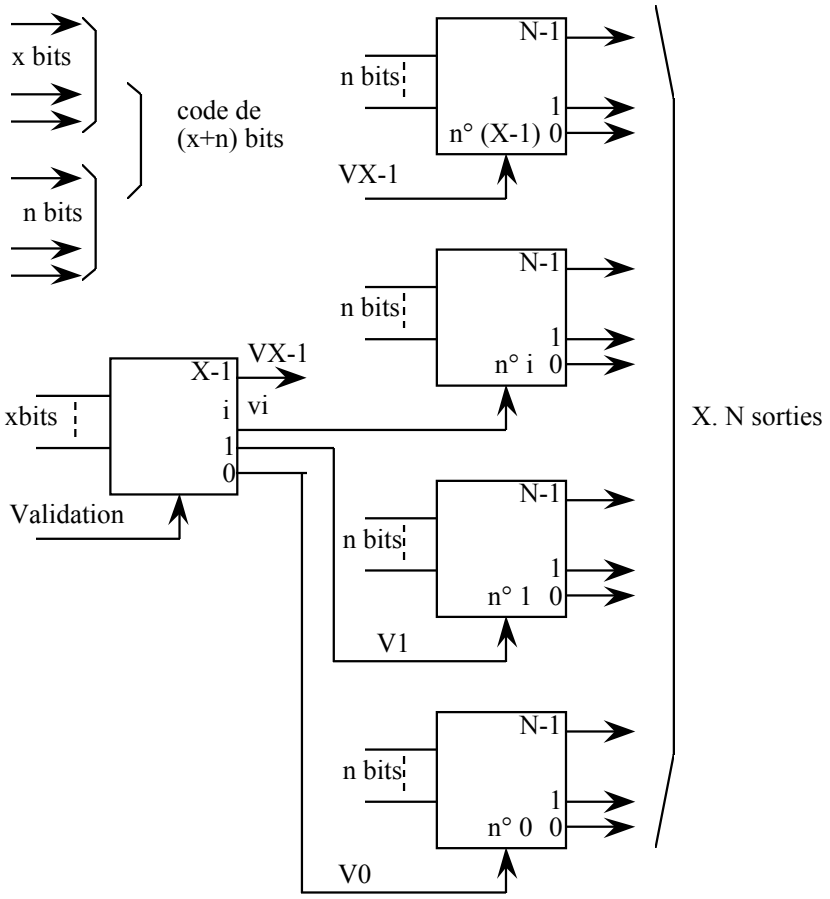


figure 4.9 : décodage de 1 parmi $X.N$ sur deux niveaux

Dans l'exemple de la figure 4.9, le dispositif, qui possède deux niveaux de décodeurs, accepte $x+n$ entrées et délivre $X.N$ sorties, avec $X=2^x$ et $N=2^n$. Le premier niveau est constitué d'un décodeur de type 1 parmi *X*. Celui-ci reçoit *x* bits en entrées et fournit sur une des sorties $V0$ à $VX-1$ un signal de sélection qui est utilisé pour valider un des *X* décodeurs du second niveau. Ces *X* décodeurs reçoivent en entrée les *n* bits restants et possèdent chacun *N* sorties. Le signal *Validation* appliqué sur le décodeur de premier niveau permet d'autoriser ou non le décodage.

2.3.5 Applications des décodeurs

Les applications des décodeurs sont nombreuses. Nous avons choisi de les illustrer à travers trois exemples.

- **<u>Exemple 1</u> : décodage d'adresse d'une mémoire ou sélection de composants dans une carte microprocesseur**

Dans une carte mère, qui possède en général un microprocesseur et plusieurs composants annexes (RAM, ROM, REPROM, boîtiers d'entrées-sorties, etc.), la fonction de décodage est essentielle afin de ne valider qu'un seul boîtier à la fois. Les signaux qui servent au décodage, et donc à la sélection des boîtiers sont issus du bus d'adresse et du bus de contrôle du microprocesseur. La figure 4.10 montre un synoptique simplifié d'une telle application des décodeurs.

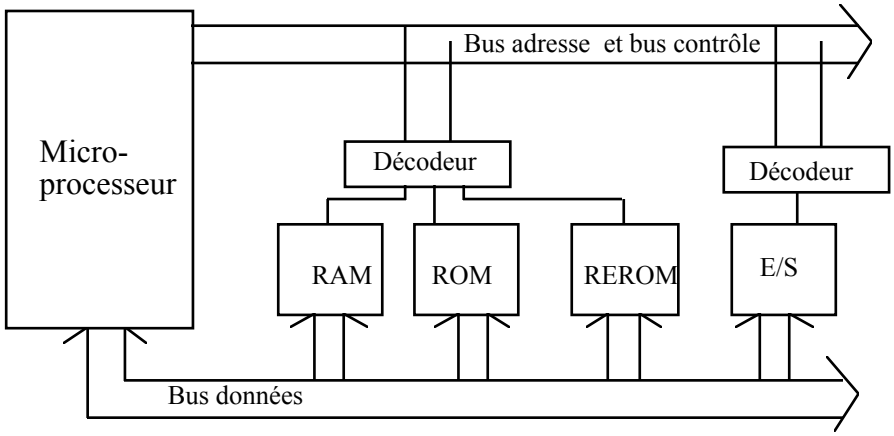


figure 4.10 : sélection d'un composant par décodeur

- **<u>Exemple 2</u> : générateur de fonction**

Nous avons vu dans la section 2.3.1.1 que les sorties Si d'un décodeur 3 vers 8 étaient régies par les équations suivantes :

$$S0 = \overline{C}\,\overline{B}\,\overline{A}$$
$$S1 = \overline{C}\,\overline{B}\,A$$
$$\dots$$
$$S7 = C\,B\,A$$

On remarque que chaque sortie Si représente un des mintermes des variables *A*, *B*, et *C*. En réunissant ces mintermes on peut donc calculer n'importe quelle fonction de trois variables à l'aide

d'un décodeur 3 vers 8 associé à des portes OU. Il est cependant à noter que l'utilisation de décodeurs en générateur de fonctions est en pratique peu fréquente.

- **Exemple 3 : démultiplexage**

Le décodeur est parfois utilisé pour **démultiplexer** des informations. Dans ce type d'application, son rôle consiste à aiguiller une information présente sur une entrée vers une sortie choisie parmi les *N* disponibles.

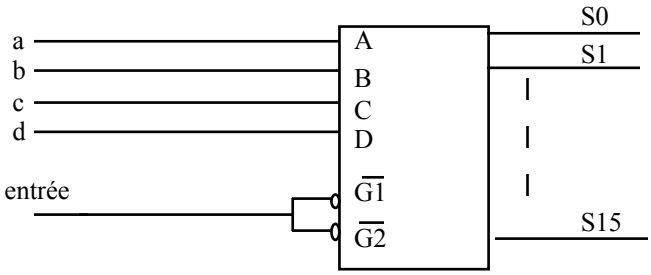


figure 4.11 : décodeur *XX* 154 utilisé pour démultiplexer

Dans l'exemple de la figure 4.11, les informations multiplexées sont injectées dans les entrées de validation ($\overline{G1}$ et $\overline{G2}$) du décodeur 4 vers 16. Les entrées *A*, *B*, *C*, et *D* possèdent ici un rôle d'entrées d'adressage. Elles servent à aiguiller l'information présente sur $\overline{G1}$ et $\overline{G2}$ vers une des 16 sorties *Si* .

N. B. Les sorties du décodeur *XX* 154 sont actives au niveau bas (à 0).

2.4 Les transcodeurs

Ces opérateurs permettent de convertir un nombre écrit dans un code C1 vers un code C2

2.4.1 Exemple 1 : le transcodeur BCD/7 segments

Le transcodeur BCD/7 segments permet de commander un afficheur alphanumérique possédant 7 segments (des diodes électroluminescentes, par exemple). Cet afficheur permet la visualisation des chiffres 0 à 9 codés en binaire naturel sur 4 bits *D*, *C*, *B*, et *A*, où *A* représente le bit de poids le plus faible.

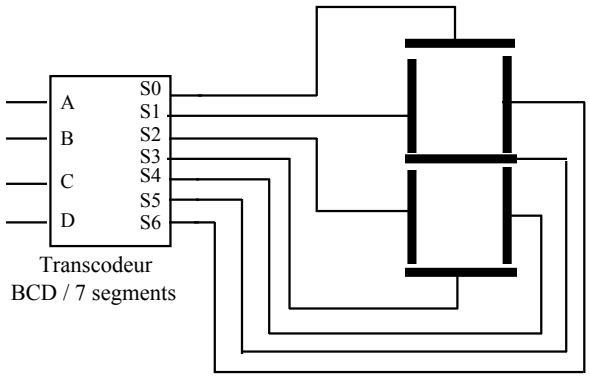


figure 4.12 : commande d'un afficheur par transcodeur

Les équations logiques du transcodeur de la figure 4.12 s'établissent sans difficultés à l'aide de 7 tableaux de Karnaugh :

$$S0 = \overline{A}.\overline{C} + A.C + B + D$$
$$S1 = \overline{A}.C + \overline{A}.\overline{B} + \overline{B}.C + D$$
$$S2 = \overline{A}.\overline{C} + \overline{A}.B$$
$$S3 = A.\overline{B}.C + \overline{A}.\overline{C} + \overline{A}.B + B.\overline{C} + D$$
$$S4 = A + \overline{B} + C$$
$$S5 = \overline{B}.C + \overline{A}.B + B.\overline{C} + D$$
$$S6 = A.B + \overline{C}$$

Ces transcodeurs existent sous la forme de circuits standard proposés par les fabricants sous les références *XX* 46 à *XX* 49. Ces circuits possèdent des entrées de contrôle supplémentaires : extinction de l'affichage des segments, allumage de tous les segments, etc. D'autres types de transcodeurs sont capables de commander des afficheurs alphanumériques. Les fonctions de trancodages sont parfois intégrées dans les afficheurs.

2.4.2 Exemple 2 : les convertisseurs Gray/binaire et binaire/Gray

Le code de Gray ou code binaire réfléchi (cf. chapitre 1, section 2.5.2.2) est largement utilisé dans les systèmes numériques, notamment dans les capteurs de position (pour coder des positions angulaires, par exemple). En effet, un seul bit change entre deux positions successives, et les risques d'informations parasites lors des transitions sont éliminés.

Les équations logiques d'un convertisseur de code Gray/binaire ou binaire/Gray s'établissent sans problème à l'aide de la méthode de Karnaugh à partir de la table de vérité suivante :

Décimal	Binaire				Gray			
	B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

tableau 4.7 : passage du code binaire au code Gray sur 4 bits

Pour des mots codés sur *n* bits, on obtient :

- **Convertisseur Gray/binaire**

$$B_n = G_n$$

et

$$B_{n-i} = G_n \oplus G_{n-1} \oplus \cdots \oplus G_{n-i} \text{ pour } i = 1,\ldots,n-1$$

- **Convertisseur binaire/Gray**

$$G_i = B_i \oplus B_{i+1} \text{ pour } i = 1,\ldots,n-1$$

et

$$G_n = B_n$$

3. Les opérateurs d'aiguillage

On distingue deux classes d'opérateurs d'aiguillage : les **multiplexeurs** et les **démultiplexeurs**.

3.1 Les multiplexeurs

Ce sont des opérateurs à $N = 2^n$ entrées d'information ou de données, $E_0, E_1, \cdots, E_{N-2}, E_{N-1}$, n entrées de sélection ou d'adressage $A_{n-1}, A_{n-2}, \cdots, A_0$, et une sortie S. L'affichage d'une adresse permet de sélectionner une entrée de données parmi N, pour l'aiguiller vers la sortie S.

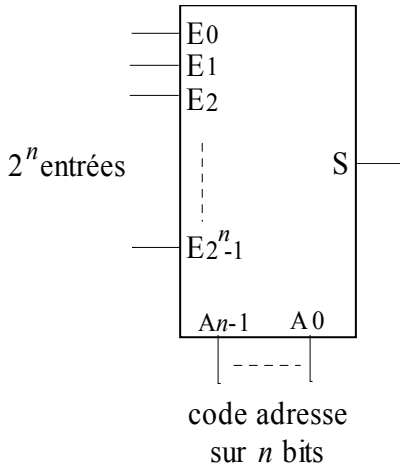


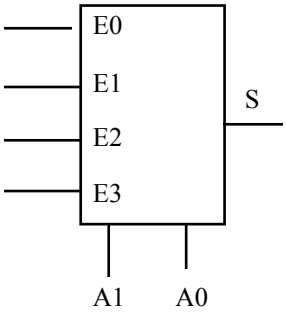
figure 4.13 : multiplexeur 2^n vers 1 ou $2^n : 1$

La fonction logique réalisée par le multiplexeur est régie par l'équation suivante :

$$S = \sum_{i=0}^{2^n-1} m_i E_i$$

où m_i est le $i^{\text{ème}}$ minterme des variables logiques $A_{n-1}, A_{n-2}, \cdots, A_0$. Par exemple, $m_0 = \overline{A_{n-1}}.\overline{A_{n-2}} \cdots \overline{A_1}.\overline{A_0}$, $m_1 = \overline{A_{n-1}}.\overline{A_{n-2}} \cdots \overline{A_1}.A_0$, …

La représentation symbolique usuelle et l'équation de sortie d'un multiplexeur 4 vers 1 sont donnés par la figure 4.14.



$$S = \overline{A_1}.\overline{A_0}.E_0 + \overline{A_1}.A_0.E_1 + A_1.\overline{A_0}.E_2 + A_1.A_0.E_3$$

figure 4.14 : multiplexeur 4 vers 1

En général, les opérateurs de multiplexage disposent d'une entrée supplémentaire de validation *V* qui permet de valider ou d'inhiber la sortie *S*. Cette entrée est souvent utilisée afin d'augmenter les capacités de multiplexage à partir d'une fonctionnalité de base. Cet aspect est détaillé dans la section suivante.

3.1.1 Exemples de multiplexeurs

La figure 4.15 présente la réalisation d'un multiplexeur 2 vers 1 à l'aide de deux interrupteurs CMOS et d'un inverseur élémentaire.

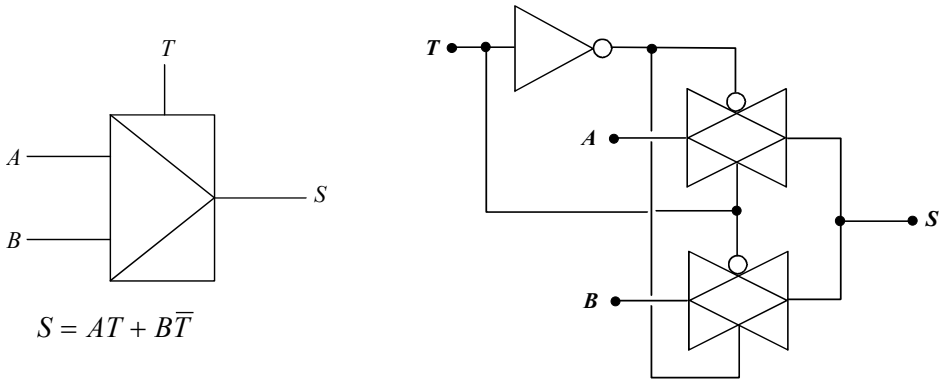


figure 4.15 : représentation usuelle et réalisation en logique CMOS d'un multiplexeur 2 vers 1

Dans l'exemple de la figure 4.16, deux multiplexeurs élémentaires 8 vers 1 avec entrées de validation sont associés afin d'obtenir un dispositif capable d'aiguiller 1 entrée parmi 16 vers la sortie. On remarque que le fil d'adresse A_3 joue un rôle particulier. Si $A_3 = 0$, le multiplexeur **1** est validé, et le multiplexeur **2**, qui correspond aux 8 bits de poids forts, est inhibé (sa sortie est forcée à 0). Les adresses A_0 à A_2 permettent donc de choisir une des 8 entrées du premier multiplexeur. Lorsque le signal A_3 vaut 1, le premier multiplexeur est inhibé (sortie à 0) et le second validé. Une porte OU permet de recueillir la sortie du multiplexeur validé.

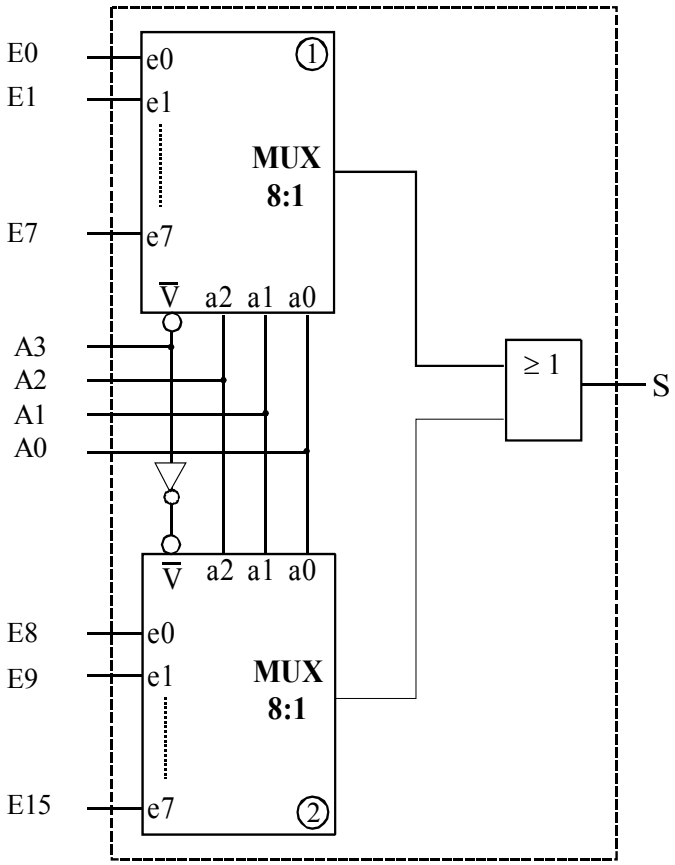


figure 4.16 : réalisation d'un multiplexeur 16 vers 1 à partir de deux multiplexeurs 8 vers 1

3.1.2 Multiplexage de mots

Certains types de multiplexeurs travaillent non pas sur des bits mais sur des mots de *x* bits. Les *x* bits qui forment le mot désigné par l'adresse sont traités simultanément et aiguillés vers les *x* bits de la sortie.

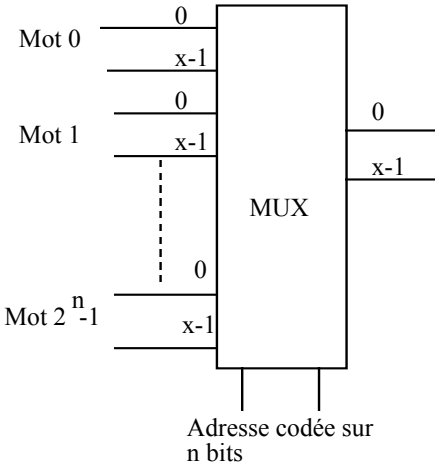


figure 4.17 : multiplexeur de 2^n mots de *x* bits

3.1.3 Applications des multiplexeurs

- **Conversion parallèle/série**

La fonction première du multiplexeur est d'aiguiller des informations provenant de *N* canaux vers un canal unique. Ce dispositif permet donc de convertir une information présente en parallèle sur les entrées d'information en une information de type série, si toutes les combinaisons d'adresses sont énumérées sur les entrées de sélection.

- **Génération de fonctions**

Les multiplexeurs peuvent également, tout comme les décodeurs, être utilisés pour la génération de fonctions logiques. En effet, toute fonction logique peut se décomposer sous la forme d'une somme de mintermes (cf. chapitre 2, section 3.1) :

$$\mathrm{f}(x_{n-1},\ldots x_1,x_0)=\sum_{i=0}^{2^n-1} d_i.m_i$$

où m_i est le $i^{\text{ème}}$ minterme des variables logiques $x_{n-1},\ldots,x_1,x_0$ et $d_i = 0$ ou 1.

Cette expression peut être rapprochée de l'équation de fonctionnement du multiplexeur. En effet, les mintermes m_i peuvent être identifiés avec les entrées d'adresse d'un multiplexeur. Les coefficients d_i sont identifiés avec les entrées d'information.

L'exemple du tableau 4.8 et de la figure 4.18 illustre la réalisation d'une fonction de deux variables à l'aide d'un multiplexeur 4 vers 1. Le passage de la table vérité au montage est immédiat.

A	*B*	F(*A*,*B*)
0	0	1
0	1	0
1	0	1
1	1	1

tableau 4.8 : table de vérité de la fonction F à réaliser

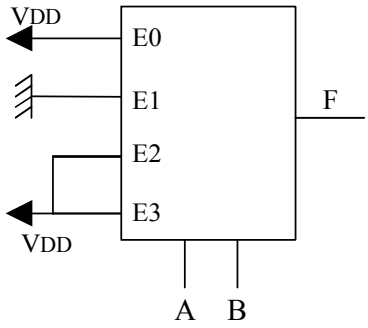


figure 4.18 : réalisation de la fonction F à l'aide d'un MUX 4 : 1

La réalisation de fonctions logiques à l'aide de multiplexeurs est en pratique utilisée dans certaines familles de circuits programmables (cf. chapitre 7). Par exemple, ACTEL propose une famille de circuits FPGA (Field Programmable Gate Arrays) dans lesquels toutes les fonctions combinatoires sont réalisées à base de multiplexeurs.

3.2 Les démultiplexeurs

Ces opérateurs, comme leur nom l'indique, réalisent la fonction inverse des multiplexeurs. Ils possèdent une entrée d'information ou de données, n entrées de sélection, et $N = 2^n$ sorties. L'affichage d'une adresse sur les entrées de sélection permet de sélectionner la sortie vers laquelle l'entrée sera aiguillée. Le démultiplexeur peut, tout comme le multiplexeur, être doté d'une entrée de validation des sorties.

3.2.1 Exemples de démultiplexeurs

En pratique, les fabricants de circuits intégrés standard proposent dans leurs catalogues des circuits pouvant être utilisés en tant que décodeurs ou en tant que multiplexeurs (cf. références *XX* 137, *XX* 138, *XX* 139). A titre d'exemple, la table de vérité du tableau 4.9 et le schéma de la figure 4.19 illustrent le fonctionnement d'un décodeur/démultiplexeur 1 vers 4. Les circuits intégrés *XX* 139 comportent deux décodeurs/démultiplexeurs de ce type.

Entrées			Sorties			
Validation	Sélection					
$\overline{G}$	B	A	$Y0$	$Y1$	$Y2$	$Y3$
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

tableau 4.9 : table de vérité d'un décodeur/démultiplexeur 1 vers 4 du circuit de référence *XX* 139

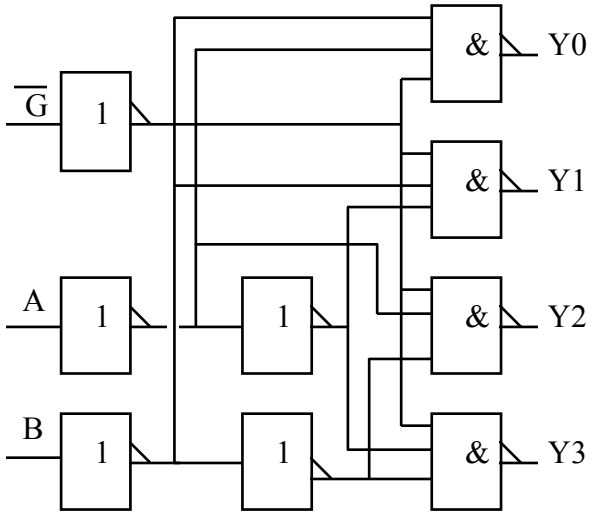


figure 4.19 : structure interne d'un décodeur/démultiplexeur du circuit *XX* 139

Lorsque ce circuit est utilisé en décodage, l'entrée $\overline{G}$, active à 0, est utilisée pour valider ou non les sorties. Dans ce mode de fonctionnement, les sorties sont également actives à la valeur logique 0. En mode démultiplexage, *A* et *B* sont les entrées d'adresse et $\overline{G}$ joue le rôle d'entrée d'information.

3.2.2 Applications des démultiplexeurs

Une des applications les plus classiques du démultiplexeur est la conversion d'une information présente en série en une information de type parallèle.

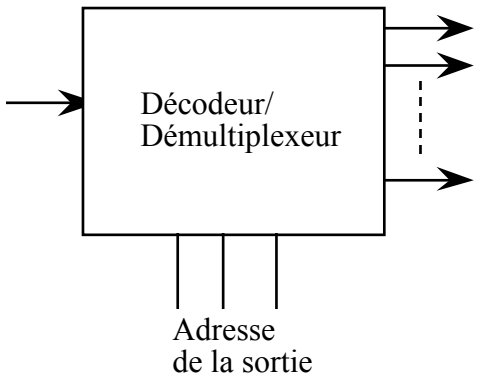


figure 4.20 : conversion série/parallèle

4. Les opérateurs de comparaison

4.1 Définition

On désigne par **opérateur de comparaison** ou **comparateur** un opérateur capable de détecter l'égalité ou l'inégalité de 2 nombres (comparateur simple) et éventuellement d'indiquer le plus grand ou le plus petit (comparateur complet). Le tableau 4.10 donne le résultat de la comparaison de 2 éléments binaires.

A	B	$A = B$	$A > B$	$A < B$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

tableau 4.10 : résultat de la comparaison de 2 bits A et B

Les équations logiques correspondantes sont les suivantes :

$$(A = B) = \overline{A \oplus B}$$
$$(A > B) = A\,\overline{B}$$
$$(A < B) = \overline{A}\,B$$

Dans le cas plus général de nombres binaires, deux nombres A et B, $A = A_{n-1} \cdots A_i \cdots A_0$ et $B = B_{n-1} \cdots B_i \cdots B_0$, sont égaux si tous les bits de même rang, A_i et B_i, sont égaux. L'équation de fonctionnement du circuit comparateur simple est alors la suivante :

$$S = \overline{A_0 \oplus B_0} . \cdots . \overline{A_i \oplus B_i} . \cdots . \overline{A_{n-1} \oplus B_{n-1}}$$
$$S = \prod_{i=0}^{n-1} \overline{A_i \oplus B_i}$$

soit encore :

$$S = \overline{(A_0 \oplus B_0) + (A_i \oplus B_i) + (A_{n-1} \oplus B_{n-1})}$$
$$S = \overline{\sum_{i=0}^{n-1} A_i \oplus B_i}$$

Cette détection peut être réalisée à l'aide d'opérateurs OU exclusif et d'une fonction NOR (figure 4.21).

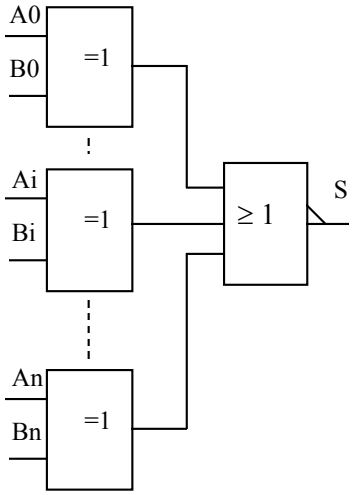


figure 4.21 : comparaison simple de 2 nombres A et B

Dans le cas d'une comparaison complète de deux nombres binaires A et B, les sorties $A < B$ et $A > B$ sont obtenues en effectuant une comparaison complète des bits de même rang, A_i et B_i, de façon prioritaire à partir des bits de poids forts (cf. tableau 4.11). Par exemple, si $A_n > B_n$ alors $A > B$, et si $A_n < B_n$ alors $A < B$. Mais si $A_n = B_n$, il est nécessaire de comparer les bits de rang $n-1$ pour décider, et ainsi de suite …

4.2 Exemple de comparateurs

Le circuit standard de référence *xx* 85 (figure 4.22) compare 2 mots de 4 bits A et B. Outre les entrées de données recevant les deux mots à comparer, il possède également trois entrées $A > B_{in}$, $A = B_{in}$ et $A < B_{in}$, permettant de cascader les comparateurs pour pouvoir comparer des nombres de plus de 4 bits. Si le comparateur est utilisé seul, les entrées $A > B_{in}$, $A = B_{in}$ et $A < B_{in}$ doivent être connectées respectivement à 0, 1, et 0. Le tableau 4.11 donne la table de vérité condensée de ce circuit.

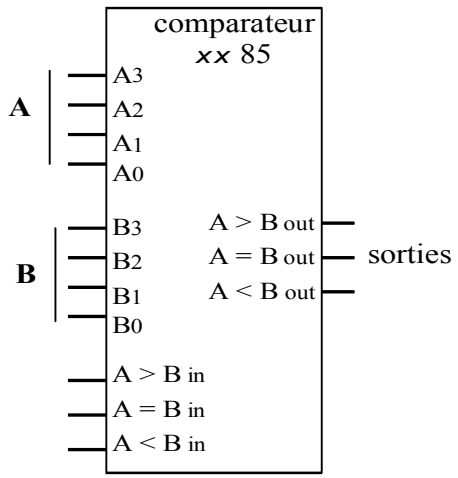


figure 4.22 : comparateur complet 4 bits *xx* 85

Entrées de données				Entrées de mise en cascade			Sorties		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A > B_{in}$	$A = B_{in}$	$A < B_{in}$	$A > B_{out}$	$A = B_{out}$	$A < B_{out}$
$A_3 > B_3$	X	X	X	X	X	X	1	0	0
$A_3 < B_3$	X	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0

tableau 4.11 : table de vérité du comparateur 4 bits XX 85

À titre d'exemple, pour comparer deux nombres de 8 bits, il suffit de relier les sorties $A > B_{out}$, $A = B_{out}$, et $A < B_{out}$ du comparateur gérant les 4 bits de poids faibles aux entrées $A > B_{in}$, $A = B_{in}$ et $A < B_{in}$ du comparateur gérant les bits de poids forts. Dans ce cas, les valeurs logiques 010 sont appliquées sur les entrées $A > B_{in}$, $A = B_{in}$ et $A < B_{in}$ du premier comparateur.

5. Les opérateurs arithmétiques

5.1 Les additionneurs

L'addition est la fonction arithmétique la plus couramment rencontrée dans les systèmes numériques. La structure des additionneurs a été intensément étudiée depuis les premières recherches sur l'architecture des calculateurs. En effet, la conception d'additionneurs rapides est importante pour effectuer des additions, et donc des multiplications et des divisions, de la manière la plus efficace possible. L'étude détaillée de tous les algorithmes de calcul arithmétique et de l'ensemble des solutions architecturales sort du cadre de ce cours. Après avoir posé le principe de base de l'addition de 2 chiffres binaires, deux architectures d'additionneurs de nombres binaires sont présentées : l'additionneur à **retenue propagée** et l'additionneur à **retenue anticipée**. Ces deux architectures correspondent à deux compromis vitesse/encombrement différents.

5.1.1 Addition de deux bits

Le **demi-additionneur binaire** prend en entrée 2 éléments binaires A_k et B_k, et délivre en sortie leur somme S_k et une retenue C_k suivant la table de vérité du tableau 4.12.

A_k	B_k	S_k	C_k
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

tableau 4.12 : table de vérité du demi-additionneur binaire

Le comportement du circuit est régi par les équations $S_k = A_k \oplus B_k$ et $C_k = A_k.B_k$. L'opérateur d'addition arithmétique binaire est donc le OU exclusif.

Le schéma de la figure 4.23 représente la structure du demi-additionneur binaire.

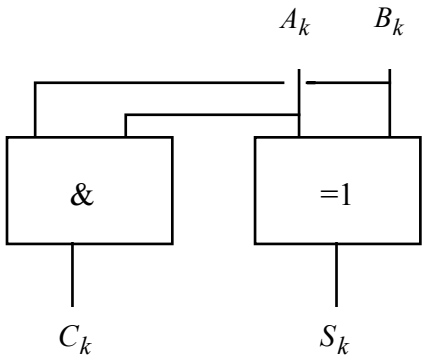


figure 4.23 : structure du demi-additionneur binaire

Pour additionner deux nombres binaires à plusieurs bits, une technique consiste à additionner successivement les bits de même poids avec la retenue de l'addition précédente. L'opérateur de base, appelé **additionneur complet**, doit alors prendre en compte une retenue entrante C_{k-1}.

A_k	B_k	retenue entrante C_{k-1}	somme S_k	retenue sortante C_k
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

tableau 4.13 : table de vérité de l'additionneur complet

Les équations logiques correspondantes peuvent s'écrire :

$$S_k = A_k \oplus B_k \oplus C_{k-1}$$
$$C_k = A_k\, B_k + (A_k + B_k)\, C_{k-1} = A_k\, B_k + (A_k \oplus B_k)\, C_{k-1}$$

L'additionneur complet peut, par exemple, être réalisé à partir de deux demi-additionneurs et d'un opérateur OU (figure 4.24).

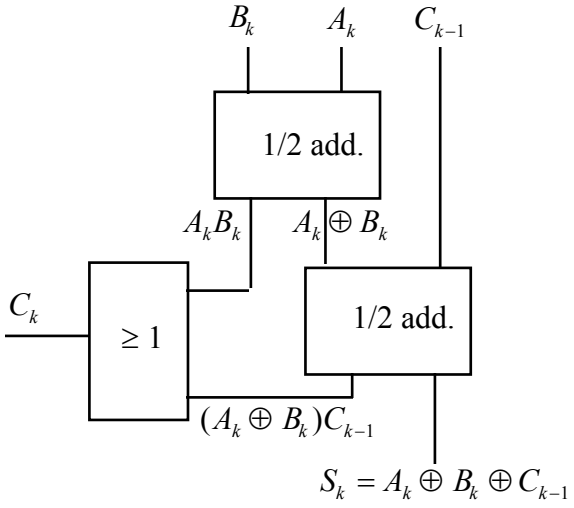


figure 4.24 : une réalisation de l'additionneur complet

5.1.2 Addition de deux nombres binaires

5.1.2.1 Additionneur à retenue propagée (ripple carry adder)

La solution la plus naturelle pour additionner deux nombres binaires A et B de n bits s'écrivant en numération de position $A_{n-1}A_{n-2}\cdots A_k\cdots A_1A_0$ et $B_{n-1}B_{n-2}\cdots B_k\cdots B_1B_0$ consiste à associer en cascade n étages d'additionneurs complets. Le résultat de l'addition est constitué de $n+1$ bits, soit n bits de somme $S_{n-1}S_{n-2}\cdots S_k\cdots S_1S_0$ et la retenue finale C_{n-1}. Chaque additionneur complet de rang k calcule la somme S_k et la retenue C_k nécessaire à l'étage suivant. La figure 4.25 présente la structure d'un additionneur de deux mots de n bits.

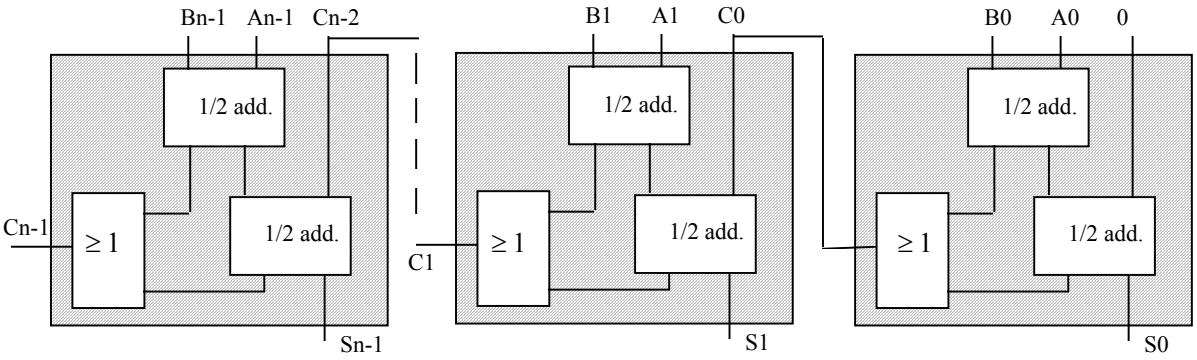


figure 4.25 : additionneur à retenue propagée ou série

Cette solution est intéressante d'un point de vue matériel car répétitive. Cependant, le résultat de l'opération n'est disponible en sortie de l'additionneur que lorsque toutes les retenues, du poids le plus faible au poids le plus fort, ont été calculées (d'où le nom d'additionneur à **retenue propagée**). Le temps de calcul du résultat est donc directement proportionnel à la taille des nombres manipulés et donc au nombre d'étages du dispositif, soit *n*. La technique de calcul anticipé de la retenue permet de remédier à ce défaut.

5.1.2.2 Additionneur à retenue anticipée (carry look-ahead adder)

Le principe de l'additionneur à retenue anticipée consiste à calculer toutes les retenues en parallèle.

L'équation logique de la retenue sortante de l'additionneur complet établie en section 5.1.1 peut s'écrire :

$$C_k = A_k B_k + (A_k \oplus B_k) C_{k-1} = G_k + P_k C_{k-1}$$

Le tableau 4.14 permet d'analyser l'obtention de la retenue sortante à partir des termes P_k et G_k.

n° de ligne	A_k	B_k	C_{k-1}	C_k	P_k	G_k	
1	0	0	0	0	0	0	Retenue sortante à 0
2	0	0	1	0	0	0	
3	0	1	0	0	1	0	Retenue entrante
4	0	1	1	1	1	0	propagée vers
5	1	0	0	0	1	0	la sortie
6	1	0	1	1	1	0	
7	1	1	0	1	0	1	Retenue sortante à 1
8	1	1	1	1	0	1	

tableau 4.14 : table de vérité de la retenue pour un additionneur complet

On observe que :

- Lignes 1 et 2 : la retenue sortante C_k est à 0,
- Lignes 3 à 6 : la retenue entrante C_{k-1} est propagée vers la sortie, $C_k = C_{k-1}$. Ces 4 lignes de la table de vérité sont caractérisées par la relation $P_k = A_k \oplus B_k = 1$. P_k est ici appelé **terme de propagation** de retenue.
- Lignes 7 à 8 : une retenue sortante à 1 est délivrée en sortie. Ces 2 lignes sont caractérisées par la relation $G_k = A_k . B_k = 1$. G_k est appelé **terme de génération** de retenue. C'est encore l'expression de la retenue R_k en sortie d'un demi-additionneur (figure 4.23).

Dans le cas de l'addition de 2 nombres *A* et *B* de *n* bits, le bit de retenue de rang *k* est donc donné par la relation $C_k = G_k + C_{k-1} P_k$.

Le principe des additionneurs à retenue anticipée consiste à calculer :

- les couples (P_k, G_k) à l'aide de demi-additionneurs, pour $0 \le k \le n-1$,
- les retenues C_k, pour $0 \le k \le n-1$, avec $C_k = G_k + C_{k-1}P_k$, directement à partir des différents termes de propagation et de génération, et de la retenue entrante du premier étage C_{-1} : $C_0 = G_0 + C_{-1}P_0$, $C_1 = G_1 + C_0P_1 = G_1 + G_0P_1 + C_{-1}P_0P_1$, etc.
- les bits de somme $S_k = P_k \oplus C_{k-1}$, pour $0 \le k \le n-1$.

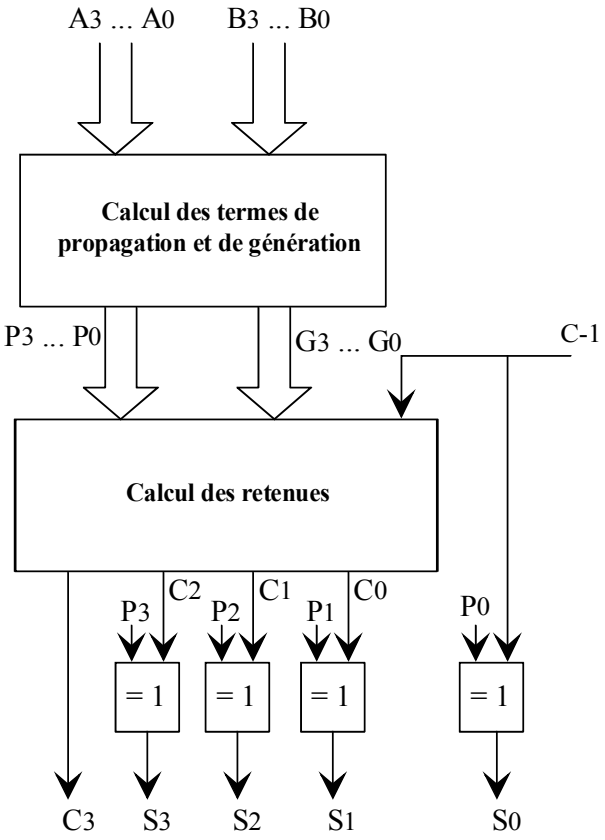


figure 4.26 : structure d'un additionneur de mots de 4 bits à retenue anticipée

Cette solution architecturale mène à des circuits plus rapides que les additionneurs à retenue propagée, car le nombre de blocs logiques traversés pour le calcul des sorties de l'additionneur est indépendant de la taille des mots en entrée. Néanmoins, ce type d'additionneur est plus encombrant que l'additionneur à retenue propagée, la logique de génération des retenues étant plus complexe. Les constructeurs proposent des circuits connus sous le nom de CLU *(Carry Look-ahead Unit)* qui assurent le calcul des termes précédents (ex : référence *XX* 182).

Les solutions étudiées ne permettent pas de traiter le cas de l'addition de deux nombres de signes différents. Ce problème est lié à celui de la soustraction qui est traitée dans le paragraphe suivant.

5.2 Les soustracteurs

Pour la soustraction $A - B$, on peut adopter la même approche que pour l'addition. On commence par définir l'opérateur binaire de base et on l'utilise pour réaliser des soustractions de nombres binaires. En pratique, se pose le problème de la représentation des nombres signés dans le cas où $B > A$. Pour résoudre ce problème, on convient d'une représentation des nombres négatifs (cf. chapitre 1), et la soustraction est alors ramenée à une addition. La représentation généralement utilisée est celle du complément vrai ou complément à 2.

5.3 Les multiplieurs et diviseurs

Dans les années 70-80, les opérations de multiplication et de division étaient effectuées de manière algorithmique dans les processeurs. Ces opérateurs sous maintenant intégrés sous forme matérielle pour des raisons de rapidité.

Le principe de la multiplication en base 2 a été présenté au chapitre 1, section 3.2.2. Il existe deux types de solutions pour la réalisation des opérateurs correspondants.

- Solution combinatoire : réseau de portes ET et d'additionneurs binaires permettant de réaliser simultanément les opérations élémentaires de la multiplication. La figure 4.27 donne une réalisation possible d'un multiplieur combinatoire de deux nombres *A* et *B* de 4 bits. Cette structure transcrit directement sous forme de circuit le réalisation « à la main » du calcul.

- Solution séquentielle : cette solution sort du cadre de ce chapitre car non combinatoire. La multiplication est réalisée en plusieurs étapes faisant intervenir des opérations successives d'addition et de décalage (cf. chapitre 5) réutilisant les mêmes opérateurs. L'implantation de cette solution est beaucoup moins encombrante que la précédente, mais présente des performances de vitesse dégradées, car elle nécessite plusieurs étapes de calcul.

La réalisation de diviseurs est en fait plus délicate que celle des multiplieurs mais les divisions sont en pratique beaucoup moins courantes dans les systèmes numériques que les additions/soustractions et les multiplications. L'approche la plus simple, qui consiste à implanter la division sous forme d'additions/soustractions et de décalages, conduit à des performances médiocres en terme de vitesse de calcul et n'est guère utilisée.

Nous ne détaillerons pas davantage les familles de circuits arithmétiques. De nombreuses architectures existent : pipe-line, parallèle, série-parallèle, cellulaires, etc. Leur étude, qui sort du cadre de ce cours, est détaillée dans [Mul89].

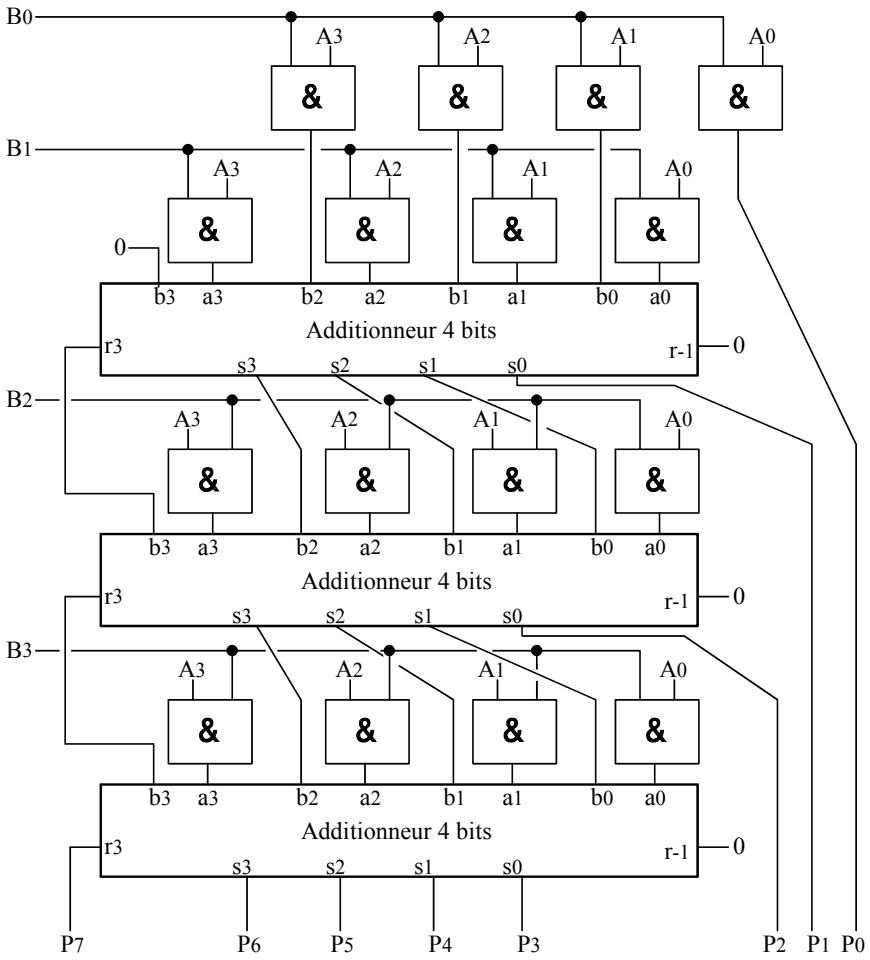


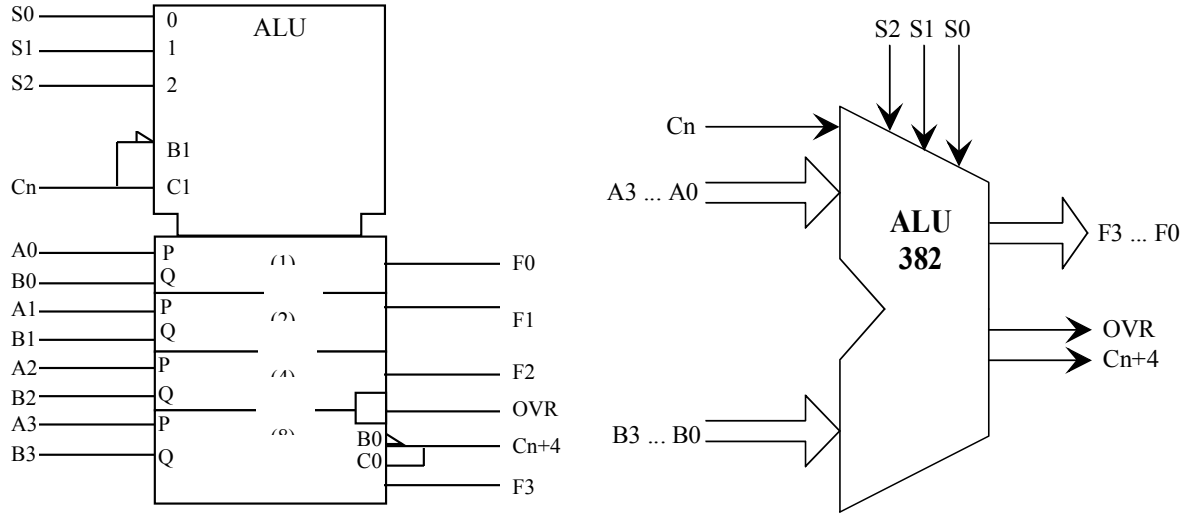
figure 4.27 : réalisation d'un multiplieur combinatoire de 2 mots de 4 bits

5.4 Les unités arithmétiques et logiques

Les fabriquants de circuits intégrés proposent de nombreux composants (sous forme de circuits standard ou bien de macrofonctions) capables d'effectuer un ensemble d'opérations arithmétiques (addition, soustraction, ...) et logiques (ET, OU, OU exclusif, …). L'opération à effectuer est déterminée par des entrées de commande ou de sélection. Ces opérateurs, appelés UAL (Unité Arithmétique et Logique) ou ALU (Arithmetic and Logic Unit), sont présents dans de nombreux dispositifs de calcul comme les microprocesseurs.

5.4.1 Exemple : le circuit intégré de référence *XX* 382

L'UAL référencée *XX* 382 est un circuit qui est constitué d'environ 80 portes logiques élémentaires. A l'aide des entrées de sélection $S0$ à $S2$, l'utilisateur peut choisir une opération parmi les huit disponibles. Cette unité accepte en entrée des mots de 4 bits.



représentation normalisée IEEE

représentation usuelle

figure 4.28 : unité arithmétique et logique *XX* 382

Le tableau 4.15 présente une table de vérité partielle de l'UAL de référence *XX* 382. Seuls les cas où *A* et *B* sont égaux à 0000 ou 1111 sont traités à titre d'exemple.

Pour les modes arithmétiques, « *A* plus *B* », « *A* moins *B* », et « *B* moins *A* », les nombres *A* et *B* sont codés en complément à 2. Dans les trois cas, l'opérateur utilisé est un additionneur dont la retenue entrante est C_n.

- $S_2S_1S_0 = 011$, « *A* plus *B* » :

  Les nombres *A* et *B* sont directement appliqués en entrée de l'additionneur. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est en fait $F = A + B + 1$, sinon $F = A + B$ (+ désigne ici l'addition).

- $S_2S_1S_0 = 001$, « *B* moins *A* » :

  Le complément à 1 de *A*, $\overline{A}$, est calculé et additionné à *B*. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = B + \overline{A} + 1 = B - A$, sinon $F = B + \overline{A} = B - A - 1$ (– désigne ici la soustraction).

- $S_2S_1S_0 = 010$, « *A* moins *B* » :

  Le complément à 1 de *B*, $\overline{B}$, est calculé et additionné à *A*. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = A + \overline{B} + 1 = A - B$, sinon $F = A + \overline{B} = A - B - 1$.

Dans ces trois modes arithmétiques, la sortie C_{n+4} est activée ($C_{n+4} = 1$) lorsque l'addition génère une retenue. La sortie *OVR* indique un dépassement de capacité de l'additionneur. Cette situation n'est pas considérée pas dans le tableau 4.15. A titre d'exemple, dans le mode « *A* plus *B* », la sortie *OVR* est activée si $A = B = 0111$.

Entrées de sélection $S2\,S1\,S0$	Opération arithmétique/logique	C_n [1]	$A =$ $A3\,A2\,A1\,A0$	$B =$ $B3\,B2\,B1\,B0$	$F =$ $F3\,F2\,F1\,F0$	OVR	C_{n+4}
000	Mise à 0 (*Clear*)	X	XXXX	XXXX	0000	—	—
001	*B* moins *A*	0	0000	0000	1111	0	0
			0000	1111	1110	0	1
			1111	0000	0000	0	0
			1111	1111	1111	0	0
		1	0000	0000	0000	0	1
			0000	1111	1111	0	1
			1111	0000	0001	0	0
			1111	1111	0000	0	1
010	*A* moins *B*	0	0000	0000	1111	0	0
			0000	1111	0000	0	0
			1111	0000	1110	0	1
			1111	1111	1111	0	0
		1	0000	0000	0000	0	1
			0000	1111	0001	0	0
			1111	0000	1111	0	1
			1111	1111	0000	0	1
011	*A* plus *B*	0	0000	0000	0000	0	0
			0000	1111	1111	0	0
			1111	0000	1111	0	0
			1111	1111	1110	0	1
		1	0000	0000	0001	0	0
			0000	1111	0000	0	1
			1111	0000	0000	0	1
			1111	1111	1111	0	1
100	$A \oplus B$	X	0000	0000	0000	—	—
			0000	1111	1111		
			1111	0000	1111		
			1111	1111	0000		
101	*A* OU *B*	X	0000	0000	0000	—	—
			0000	1111	1111		
			1111	0000	1111		
			1111	1111	1111		
110	*A* ET *B*	X	0000	0000	0000	—	—
			0000	1111	0000		
			1111	0000	0000		
			1111	1111	1111		
111	Mise à 1 (*Preset*)	X	XXXX	XXXX	1111	—	—

tableau 4.15 : table de fonctionnement partielle de l'UAL XX 382

Cette UAL permet également d'effectuer 3 opérations logiques sur les mots *A* et *B* : OU exclusif, OU et ET. Ces trois opérations s'effectuent bit à bit. En outre, les sorties $F3F2F1F0$ peuvent être forcées à 0 ou à 1. Les sorties C_{n+4} et *OVR* ne sont, *a priori*, utilisées que dans les modes arithmétiques, leurs valeurs ne sont donc pas données dans les modes logiques car sans intérêt.

[1] C_n : entrée de retenue pour l'addition et entrée de retenue inversée pour la soustraction

6. Bibliographie

[BH90] J.-M. Bernard et J. Hugon, *Pratique des circuits logiques,* Collection technique et scientifique des télécommunications, Eyrolles, 1990.

[Mot] http://www.motorola.com/

[Mul89] J.-M. Muller, *Arithmétique des ordinateurs, opérateurs et fonctions élémentaires*, Etudes et recherches en informatique, Masson, 1989.

[NS] http://www.national.com/

[TI] http://www.ti.com/