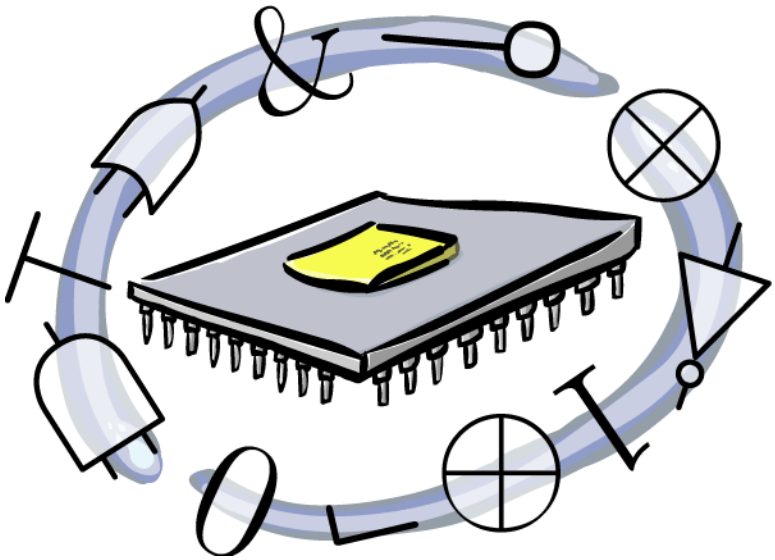
Institut Universitaire de Technologie de CRETEIL-VITRY
Département de Génie Électrique et Informatique Industrielle

SIN1 - SYSTEME D'INFORMATION NUMERIQUE

COURS
Du codage des nombres à la logique séquentielle



Année universitaire 2013-2014
J. VILLEMEJANE - julien.villemejane@u-pec.fr

M 1102 — **Système d'information numérique**

SIN1 — *Coeff 3*

60h15	CM 13h45 *11 x 1h15*	TD 22h30 *15 x 1h30 dont 2 DS*	TP 24h *8 x 3h dont 1 Test*
AA	18h	20h	22h
	1 Introduction aux systèmes numériques *Nécessité du codage des informations*		
	2 Codage de l'information *Caractères* *Entiers naturels et relatifs*	1 Codage de l'information *Entiers naturels et relatifs* *Addition et multiplication*	
		2 Codage des entiers relatifs	0 Découverte de l'environnement et de la carte
	3 Représentation des systèmes combinatoires *Algèbre de Boole – équations* *Table de vérité* *Portes logiques fondamentales et dérivées*	3 Formulation d'un problème logique 4 Représentation des systèmes combinatoires *Mise en équation* *Table de vérité*	1 Systèmes combinatoires simples
	4 Simplification d'équations *Méthode analytique* *Méthode graphique (Karnaugh)*	5 Simplification analytique et graphique d'équations	
	5 Fonctions combinatoires standard *Multiplexeurs* *Décodeurs* *Fonctions arithmétiques*	6 Fonctions combinatoires standard 7 Fonctions combinatoires standard 2	
	6 Représentation comportementale *Introduction au VHDL*		2 Fonctions combinatoires standard
		DS1 COMBINATOIRE	
	7 Introduction aux systèmes séquentiels *Notion d'état* *Bascules – Mémorisation*	8 Bascules et mémorisation	
	8 Représentation des systèmes séquentiels *Diagramme d'état* *Table de transitions* *Structure*		
	9 Fonctions séquentielles standard *Compteurs / Registres* *Synthèses classique et comportementale*	9 Synthèse de compteurs 10 Synthèse de registres	3 Synthèse de compteurs 4 Synthèse de registres
	10 Machines à état *Structure* *Conception – Synthèse*	11 Synthèse structurelle d'une machine à état	
	11 Machines à état d'un point de vue comportemental	12 Synthèse comportementale d'une machine à état	5 Machine à état simple
		13 Automate : structure et comportement	6 Automate
		DS2 SEQUENTIEL	**Test**

Objectifs du module

- Connaître les fonctions de base de l'électronique numérique
- Utiliser un langage de description matérielle des circuits
- Mettre en oeuvre des systèmes numériques

Compétences visées

- Décomposer une fonction en blocs **combinatoires** et **séquentiels**
- Choisir et mettre en oeuvre un **circuit numérique** conventionnel ou programmable
- Utiliser une **chaîne de développement**
- Décrire, programmer, simuler et tester la fonctionnalité à réaliser

Organisation du module

- 11 x 1h15 de cours (amphi)
- 13 x 1h30 de TD (groupe)
- 8 x 3h de TP (demi-groupe)
  - dont **1 TP Test** (noté ! ! coeff. 1)

Evaluation du module

- DS1 : logique combinatoire (semaine 9) - coeff 1
- DS2 : logique séquentielle (semaine 17) - coeff 1
- TP Test : langage VHDL (semaine 16) - coeff 1
- QCM1 : codage des nombres - coeff 0.25
- QCM2 : logique combinatoire - coeff 0.25

Informations utiles

Pour toute question sur le cours ou les TD/TP : **julien.villemejane@u-pec.fr**

Les cours, TD et TP au format numérique : **http ://cours.villemejane.net/**

FPGA / CPLD / langage VHDL : **http ://www.xilinx.com/**
Xilinx - Logiciel ISE WebPack (gratuit - nécessite 8 Go)

Table des matières

Cours 0 - Introduction aux systèmes numériques 6

1 Rappel sur les systèmes 6

2 Chaine d'acquisition d'informations 6

3 Systèmes numériques 6

3.1 Intérêt des grandeurs numériques . . . 6
3.2 Différents types de systèmes numériques . . . 7
3.3 Transmission de l'information . . . 7
3.4 Classement des systèmes numériques . . . 8

Cours 1 - Codage des informations 9

1 Le codage de l'information 9

1.1 Types d'informations . . . 9
1.2 Représentation de l'information . . . 9

2 Codage des caractères alphanumériques 9

2.1 Code ASCII (American Standard Code for Information Interchange) . . . 10
2.2 Unicode . . . 10

3 Codage des données numériques 11

3.1 Codage des entiers naturels . . . 11
3.2 Transcodage . . . 13
3.3 Codage des entiers relatifs . . . 13
3.4 Codage des nombres réels . . . 15
3.5 Autres codes . . . 16

Cours 2 - Systèmes combinatoires 21

1 Introduction aux systèmes combinatoires 21

2 Représentation des systèmes combinatoires 21

3 Algèbre de Boole 21

3.1 Opérateurs fondamentaux . . . 22
3.2 Opérateurs dérivés . . . 23
3.3 Propriétés des opérations . . . 24

4 Synthèse de circuits combinatoires **24**
4.1 Méthode analytique de simplification d'équations . . . 25
4.2 Méthode graphique de simplification d'équations (Karnaugh) . . . 25

5 Fonctions combinatoires standard **26**
5.1 Multiplexeurs / Démultiplexeurs . . . 27
5.2 Codeurs et décodeurs . . . 28
5.3 Fonctions arithmétiques . . . 29

Annexe : Formes canoniques des expressions logiques **34**
8.1 Somme de produits . . . 34
8.2 Produit de sommes . . . 34
8.3 Passage d'une forme canonique à une autre . . . 35
8.4 Fonctions à spécification incomplète . . . 35

Cours 3 - Systèmes séquentiels 37

1 Introduction aux systèmes séquentiels **37**
1.1 Notion d'état . . . 37
1.2 Systèmes synchrones ou asynchrones . . . 37

2 Représentation des systèmes séquentiels **37**
2.1 Chronogramme . . . 38
2.2 Table de transition . . . 39
2.3 Diagramme d'état . . . 39
2.4 Structure d'une machine à état . . . 40

3 Bascules : composants élémentaires de la logique séquentielle **40**
3.1 Bascule RS . . . 40
3.2 Bascule JK . . . 41
3.3 Bascule T . . . 41
3.4 Bascule D . . . 41

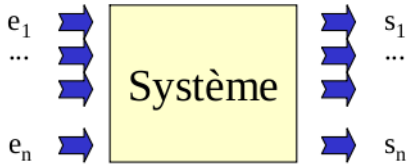
4 Fonctions séquentielles standard **42**
4.1 Compteurs / Décompteurs / Diviseurs de fréquence . . . 42
4.2 Registres . . . 45

5 Machine à états **48**
5.1 Modèles de machine à états . . . 48
5.2 Conception et synthèse de machines à états . . . 49

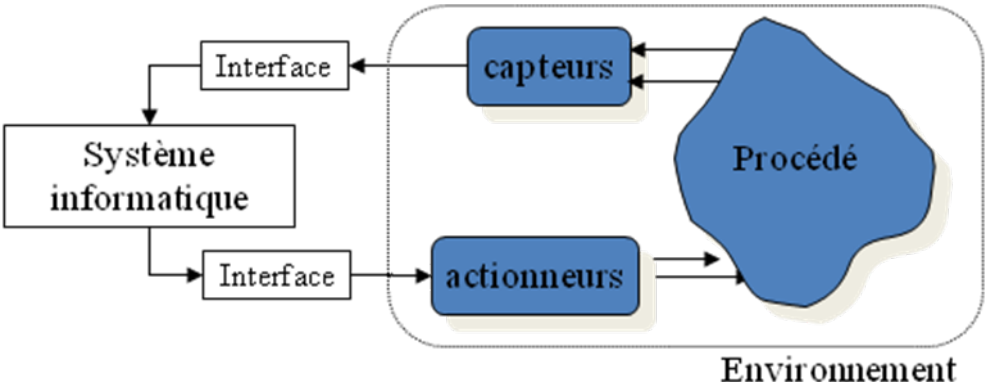
Introduction aux systèmes numériques

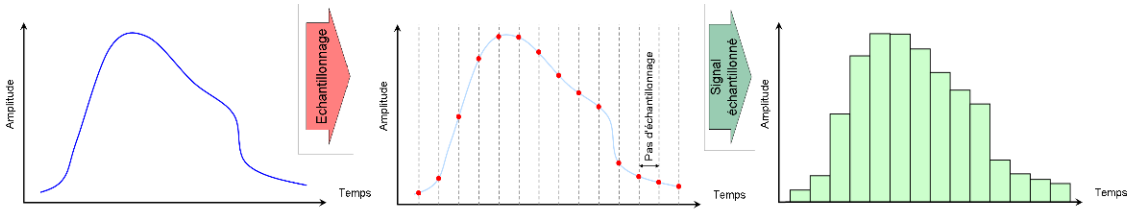
1. Rappel sur les systèmes

Un **système** est caractérisé par :
- sa **relation entrées-sorties** : $s = f(e)$
- son **temps de réponse** : T_d



2. Chaine d'acquisition d'informations





3. Systèmes numériques

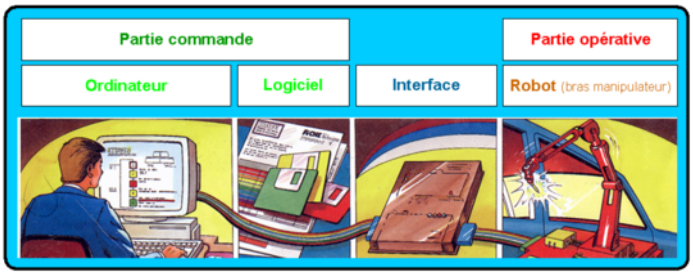
3.1. Intérêt des grandeurs numériques

Signaux	Analogiques	Numériques
Type d'informations	Analogue à la grandeur considérée	Quantification Echantillonnage
Informations	Température, Luminosité...	Données
Avantages	Au plus proche de l'environnement Capteurs	Traitement Transport d'informations
Inconvénients	Transport de l'information	Erreur de conversion

3.2. Différents types de systèmes numériques

Il existe différents types de systèmes numériques :

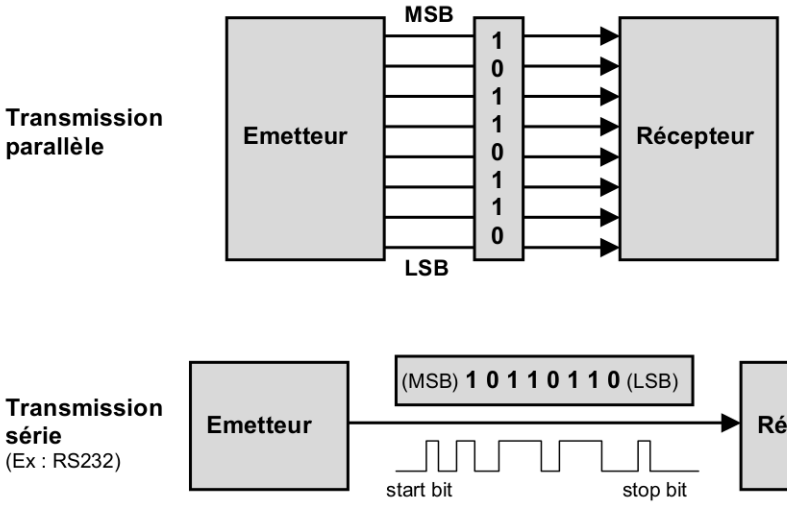
1. **Systèmes informatiques** (processeurs généralistes)
2. **Systèmes d'interface** (systèmes dédiés)
   - Capteurs intelligents (micro-controleurs)
   - Traitement d'informations (traitement du signal)
3. **Controle de systèmes spécifiques** (systèmes personnalisés et programmables)



3.3. Transmission de l'information

Quelque soit la forme de l'information récupérée par un système, sa transmission doit être faite de façon sûre et le plus rapidement possible vers d'autres systèmes.

Pour cela, il existe deux modes de transmission possible [1] :

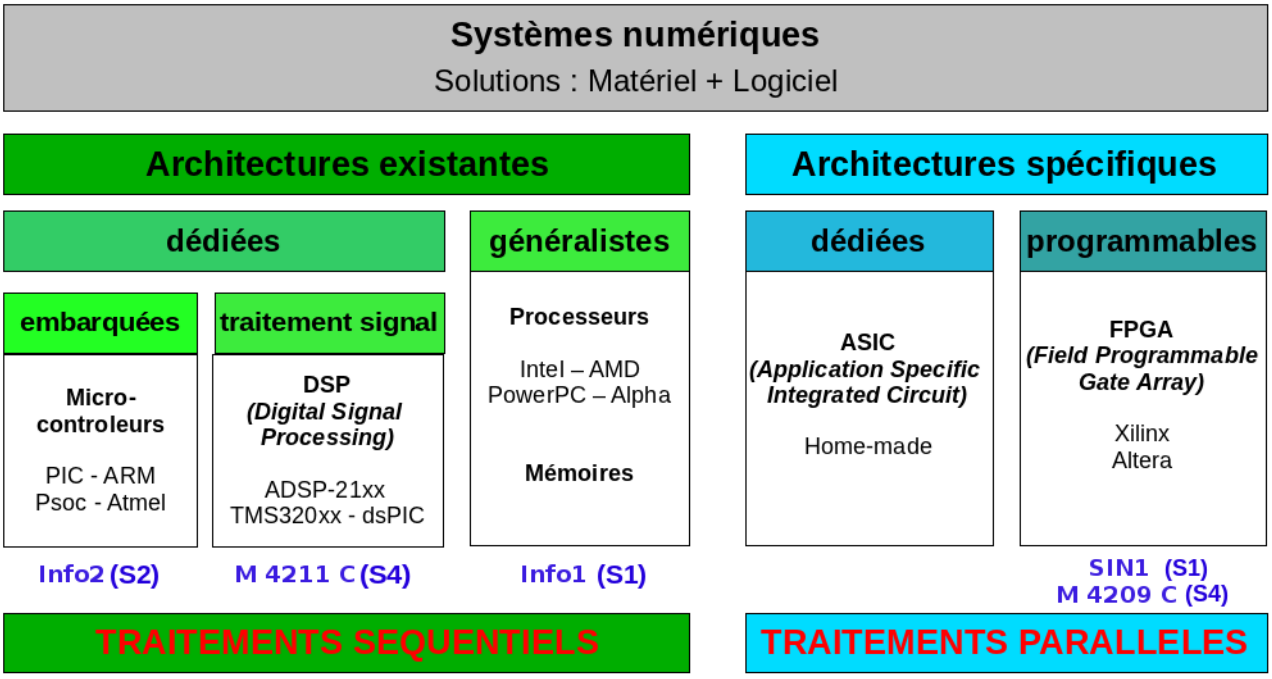


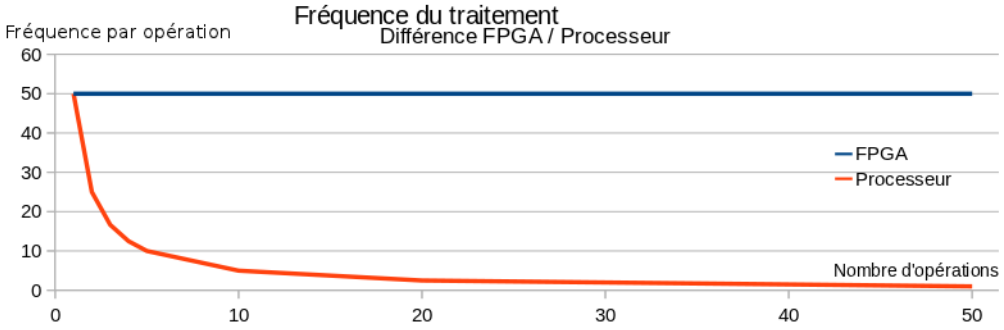
Quelque soit le cas de figure, il est indispensable que **tous les systèmes numériques** puissent comprendre les informations numériques qui transitent entre eux. Il est donc nécessaire d'adopter un même codage pour l'ensemble des données numériques.

1. L'étude de ces transmissions et les protocoles de communication associés ne font pas l'objet de ce cours, mais seront étudiés dans d'autres modules du DUT GEII.

3.4. Classement des systèmes numériques

On peut alors classer ces différents types de systèmes selon le schéma suivant.





Codage des informations

1. Le codage de l'information

Il existe différents types d'informations dans le monde du numérique : du texte, des nombres, des sons, des images, des instructions...

1.1. Types d'informations

Une information numérique peut être de *deux types* :

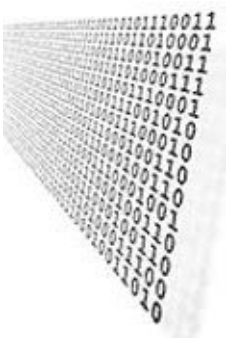
- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée**, sur laquelle des instructions pourront être faites.

Parmi les données, il existe des sous-catégories qui ne seront pas traitées de la même manière par les systèmes informatiques. Parmi ces données, on peut citer :

- des **données non numériques** (caractère alphanumérique) ;
- des **données numériques** :
  - entiers naturels (0 ; 1 ; 315 ...)
  - entiers relatifs (-1578 ; -15 ; -1 ...)
  - réels (3.1415 ; 4587.598 ...)

1.2. Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**. Afin d'avoir un langage universel, ces données sont représentées sous **forme binaire**, c'est à dire une *suite de 0 et de 1*.



L'information élémentaire est appelé **BIT** (BInary digiT).

Un **mot binaire** est un **regroupement de n bits**. Il permet d'obtenir 2^n combinaisons différentes.

Un regroupement de **8 bits** s'appelle **un octet**.

2. Codage des caractères alphanumériques

Les premiers codes ont été établis pour répondre au besoin de transmettre des `messages` rapidement en utilisant les moyens de communication disponibles à l'époque. L'alphabet a été codé au moyen de signaux à états finis : télégraphe Chappe (optique), code Morse (radio)...

Des systèmes de codage basés sur la présence ou l'absence d'un signal électrique (1 ou 0) sont apparus en même temps que l'informatique. On ne retiendra ici que les codes les plus utilisés actuellement : **ASCII** et **Unicode**.

2.1. Code ASCII (American Standard Code for Information Interchange)

Le codage **ASCII** (normalisé en 1967) représente un jeu de **128** caractères au moyen de **7** signaux à 2 états. La table de correspondance est représentée ci-dessous, sous forme rectangulaire pour être plus compacte.

Ce code comprend 26 majuscules, 26 minuscules, 10 chiffres, 32 symboles, 33 codes de contrôle et un espace. Ce codage n'est réellement adapté qu'à la langue anglaise qui est dépourvue de toute *décoration* (ou signes diacritiques : accent, cédille, tilde, etc..) sur les caractères.

Des codes de commande pour les anciens terminaux complètent cette liste : seuls quelques uns sont encore utilisés aujourd'hui (NUL, BS, HT, LF, VT, FF, CR, ESC, DEL).

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

ASCII/8859-1 Text

A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

Unicode Text

A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
س	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0010 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
≠	0010 0010 0111 0000
γ	0000 0011 1011 0011

Ce code est très limité, mais il suffit à de nombreuses activités techniques dans le domaine du traitement de l'information (systèmes d'exploitation, claviers, imprimantes, internet, etc..).

C'est le codage utilisé en **langage C** pour les variables de type `char`, codé sur 8 bits (ou 1 octet).

2.2. Unicode

Après des tentatives d'extension du code ASCII par les firmes informatiques, **Unicode** a été créé de manière concertée. Au départ, les caractères étaient codés sur 16 bits. Depuis, il a été étendu à 32 bits.

La première version a été développée en 1991. L'unicode est mis à jour régulièrement pour prendre en compte de nouveaux alphabets : 5.2 (2009), 6.0 (2012), 6.1 (2012), 6.3 (2013).

Dans la version 6.0 (janvier 2012), l'Unicode permettait de coder :

- 137 468 caractères à usage privé ;
- 109 242 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques (accent...) et signes de ponctuation ;
- plusieurs centaines de caractères de contrôle ou modificateurs spéciaux.

Pour assurer la compatibilité, les codes des caractères du jeu ASCII sont les mêmes en ASCII qu'en Unicode, il suffit d'ajouter des zéros à gauche pour arriver à 16 bits.

3. Codage des données numériques

Selon le type de données numériques à traiter, on pourra utiliser des codages différents. Le codage est réalisé à l'aide de l'algorithme de conversion associé au type de la donnée. Les opérations arithmétiques sont ensuite effectuées en arithmétique binaire.

On rappelle ici l'addition et la multiplication en binaire :

0 + 0 = 0	0 x 0 = 0
0 + 1 = 1	0 x 1 = 0
1 + 0 = 1	1 x 0 = 0
1 + 1 = 0 (et une retenue)	1 x 1 = 1

Dans tous les cas, un **code pondéré**, c'est à dire que chaque symbole d'un nombre aura un poids spécifique dans l'écriture de celui-ci, sera utilisé.

3.1. Codage des entiers naturels

3.1.1 Base décimale

La base décimale (dite base 10) possède 10 chiffres {0,1,...,9}.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre. Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

	Centaine		Dizaine		Unité
345 =	3×10^2	+	4×10^1	+	5×10^0

On peut généraliser cette équivalence par l'expression : $n = \sum_{i=0}^{p-1} d_i \cdot 10^i$ avec $d_i < 10 \; \forall i$ pour un nombre n écrit avec p chiffres décimaux d_i.

3.1.2 Autres bases

En généralisant ce qui précède, on peut utiliser d'autres bases de numération pour des besoins particuliers.
Soit B la base de numération, soient b_i les chiffres utilisés pour cette base, on a bien évidemment : $b_i < B$, $\forall i$.

La décomposition d'un nombre dans la base B se traduit par : $n = \sum_{i=0}^{q-1} b_i \cdot B^i$ avec $b_i < B \; \forall i$

$$(b_{q-1}...b_1b_0)_B = b_{q-1} \times B^{q-1} + ... + b_1 \times B^1 + b_0 \times B^0$$

Nombre minimum de symboles pour le codage dans une base d'un nombre Le nombre q minimum de symboles nécessaires pour écrire n en base B est donné par l'inégalité :

$$q \geq \frac{\ln n}{\ln B}$$

Dynamique de codage Dans les machines de traitement de l'information, les nombres sont représentés par une certaine quantité de symboles (généralement binaires). Cette quantité est fixée lors de la conception des machines. Il en resulte une certaine étendue ou *dynamique de codage* liée à ce nombre de symboles représentant l'information à coder.

Dynamique de codage des entiers positifs :

Nombre de symboles	Dynamique de codage	Base 10	Base 2
1	0 à $(B^1 - 1)$	0 à 9	0 à 1
2	0 à $(B^2 - 1)$	0 à 99	0 à $(11)_2 = (3)_{10}$
4	0 à $(B^4 - 1)$	0 à 9 999	0 à $(1111)_2 = (15)_{10}$
8	0 à $(B^8 - 1)$	0 à 99 999 999	0 à $(1111\ 1111)_2 = (255)_{10}$
16	0 à $(B^{16} - 1)$	0 à $(10^{16} - 1)$	0 à $(65535)_{10}$
...	...	...	...
k	0 à $(B^k - 1)$	0 à $(10^k - 1)$	0 à $(2^k - 1)_{10}$

3.1.3 Base binaire / Codage binaire naturel

La base binaire (dite base 2) possède 2 symboles {0,1}.

Un **nombre binaire** est une somme de termes où chacun est une **puissance de deux** multipliée par un symbole.

$$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Le symbole le plus à gauche d'un nombre et qui a le plus de poids dans l'écriture d'un nombre binaire est appelé **bit de poids fort** ou **MSB** (Most Significant Bit). De la même manière, le symbole le plus à droite et qui a le moins de poids est appelé **bit de poids faible** ou **LSB** (Less Significant Bit).

3.1.4 Base hexadécimale

La base hexadécimale (dite base 16) possède 16 symboles {0,1,...,9,A,B,C,D,E,F} (où A=10, B=11,... F=15).

Un **nombre hexadécimal** est une somme de termes où chacun est une **puissance de seize** multipliée par un symbole.

$$D7A = D \times 16^2 + 7 \times 16^1 + A \times 16^0$$

La manipulation des nombres en base 2 est une nécessité pour les informaticiens, mais c'est une tâche lourde et propice aux erreurs.
La base hexadécimale (base 16) simplifie beaucoup la manipulation des machines.

La correspondance entre un groupe de 4 chiffres en base 2 et le chiffre hexadécimal correspondant est indiqué dans le tableau ci-après :

Base 2	Base 16	Base 10
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Base 2	Base 16	Base 10
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

3.2. Transcodage

Le transcodage est le fait de **convertir un nombre** d'une base vers une autre.

3.2.1 Base B vers Base décimale

Pour passer d'une base B vers la base décimale (base la plus utilisée par l'être humain), on utilise le principe de la décomposition d'un nombre dans cette base B (vu précédemment).

Ainsi, un nombre dans une base B quelconque s'écrira :

$$(b_{q-1}...b_1b_0)_B = b_{q-1} \text{ x } B^{q-1} + ... + b_1 \text{ x } B^1 + b_0 \text{ x } B^0$$

3.2.2 Base décimale vers Base B

Pour le transcodage inverse, de la base décimale vers une base B quelconque, on divise successivement le nombre à transcoder par B. Les calculs s'arrêtent lorsque le quotient arrive à 0.

$$\begin{array}{c|c} N & B \\ \hline a_0 & N_1 \end{array} \rightarrow \begin{array}{c|c} N_1 & B \\ \hline a_1 & N_2 \end{array} \rightarrow \ ... \ \begin{array}{c|c} N_{n-1} & B \\ \hline a_{n-1} & 0 \end{array}$$

Pour obtenir la valeur convertie, il faut ensuite récupérer l'ensemble des restes de ces divisions en partant du **dernier reste obtenu**, qui correspond alors au poids fort du nombre final (a_{n-1}), **jusqu'au premier reste** calculé qui correspond au poids faible (a_0).

3.3. Codage des entiers relatifs

3.3.1 Problématique

Lors de traitements arithmétiques, il est nécessaire de disposer des nombres entiers négatifs, pour que le résultat de certaines opérations puisse être représenté. Dans le cas d'une machine ayant un certain nombre de bits, et donc une certaine dynamique de codage en binaire naturel, la solution consiste à **sacrifier une partie de cet intervalle** pour la représentation des nombres négatifs. Généralement, on partage l'intervalle en deux morceaux égaux l'un pour les nombres positifs et la valeur zéro, et l'autre pour les nombres négatifs. Il y a donc une légère dissymétrie entre les nombres positifs et négatifs, sans grande importance en pratique.

3.3.2 Dynamique de codage des entiers relatifs

Dans le cas des machines les plus courantes, les dynamiques de codage deviennent :

Nombre de bits	Dynamique de codage
4	de -8 à +7
8	de -128 à +127
16	de -32768 à +32767
32	de -2 147 483 648 à +2 147 483 647

3.3.3 Code complément à 2

Le **code complément à 2** (C2 en abrégé) est un outil permettant de **coder des nombres entiers relatifs** tout en garantissant la dynamique de codage précédente et permettant les calculs arithmétiques entre deux nombres codés ainsi.

Le codage obtenu est alors **pondéré** : le bit de poids fort représente une valeur négative alors que les autres bits représentent des valeurs positives. Ce codage n'a donc de sens que pour un **nombre de bits convenu à l'avance** : le code C2 n'est pas extensible à volonté.

On doit donc impérativement connaître (ou trouver) le nombre de bits à employer pour établir le code C2 d'un nombre N.

Dans tous les cas, le nombre sera codé de la façon suivante :

$$N = -b_{q-1} \cdot 2^{q-1} + \sum_{i=0}^{q-2} b_i \cdot 2^i$$

où b_i représente un bit (0 ou 1) de poids i.

3.3.4 Algorithme de calcul du complément à 2

Il existe une méthode systématique pour le calcul du complément à 2 d'un nombre.

Si nécessaire, rechercher le nombre minimal de bits p pour coder le nombre n en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **Si** $N \geq 0$ (nombres de 0 à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à p bits (en complétant à gauche par des 0). Le bit de poids fort est égal à 0.
- **Si** $N < 0$ (nombres de -2^{p-1} à -1) :
  1. coder $|N|$ en binaire en complétant à gauche par des 0 pour obtenir un code sur p bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter 1 au résultat (**complèment à deux** ou C2)

Une autre méthode de codage peut être utilisée pour les nombres négatifs : additionner 2^q à N, puis coder le nombre obtenu en binaire naturel.

3.4. Codage des nombres réels

3.4.1 Problématique

Nous avons vu précédemment que la **représentation binaire** d'un nombre était **limitée** sur une plage de valeurs appelée la **dynamique de codage**. Sur cet intervalle de valeurs, on pouvait alors coder tous les nombres entiers présents, car ils sont en nombre limité.

Sur l'ensemble des réels, la problématique est tout autre. En effet, sur le même intervalle que précédemment, il existe une infinité de nombres réels. La représentation binaire entrainera alors systèmatiquement une **imprécision** dans le codage.

Deux possibilités existent alors pour coder des nombres réels en binaire :
- en **virgule fixe** : la partie entière et la partie réelle seront toujours codées sur le même nombre de bits ;
- en **virgule flottante** : le nombre à coder sera ramené à un certains nombres de chiffres significatifs et un exposant (norme IEEE 754).

3.4.2 Codage en virgule fixe

On considère un dénominateur *implicite* 2^p commun à toute la représentation. Le code équivalent est généralement appelé Q_p.

Il est alors possible de représenter le nombre codé par la relation suivante :

$x =$	a_{n-1}	+	a_{n-1}	+ ... +	a_1	+	a_0
	x		x		x		x
Rang	$p^{n-1-p-1}$		p^{n-p-2}	...	2^{-p+1}		2^{-p}
Puissance	n-1-p-1		n-p-2	...	-p+1		-p

La méthode de codage d'un réel x sur n bits en code Q_p peut être décrite par :
- multiplier x par 2^p ;
- ne conserver que la partie entière du résultat précédent et la coder en code C2.

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo "grand public", la majorité des processeurs de traitement du signal étant basée sur cette technique moins couteuse que le codage en virgule flottante.

3.4.3 Codage en virgule flottante : norme IEEE754

En codant séparément des chiffres significatifs et un exposant, le tout en mode binaire, le codage en virgule flottante des nombres réels présente un énorme avantage pour les calculs scientifiques : les débordements de capacité sont bien moins fréquents, car l'étendue de l'intervalle codé est bien plus grand. On peut mener les calculs sans être obligé de surveiller les dépassements à chaque étape ! De plus, par construction, la précision relative des nombres est constante et indépendante de leur magnitude, car ils ont un nombre de bits significatifs constant.

Cette méthode assure une utilisation optimale des bits consacrés à la représentation des nombres. Cependant, la complexité des opérateurs permettant les opérations de base est incomparablement plus grande que dans le cas de la représentation en virgule fixe. Il a fallu attendre longtemps avant que les machines de traitement de l'information aient accès à des *coprocesseurs numériques* spécialisés dans ces opérations.

Régles de codage IEEE 754 simple précision (32 bits) : Un bit est réservé pour le signe S de la **mantisse** (0 pour positif, 1 pour négatif), 8 bits pour l'**exposant** E et 23 bits pour la partie fractionnaire de la mantisse F.

Bit	31	30..23	22..0
Dénomination	S	E	F

La formule permettant de trouver la valeur décimale à partir de la représentation IEEE est :

$$v = (-1)^S \cdot 2^{E-127} \cdot F$$

où E et F sont des représentations binaires et $0 < E < 255$.

Cas spéciaux :
- Si $E = 255$ et $F \neq 0$ alors v est NaN (Not a Number : code illicite) ;
- Si $E = 255$ et $F = 0$ alors v est $\pm\infty$ selon la valeur de S ;
- Si $E = 0$ et $F \neq 0$ alors v est un nombre dénormalisé dont la valeur est : $v = (-1)^S \cdot 2^{E-126} \cdot F$

Il existe également un format double précision (64 bits).

3.5. Autres codes

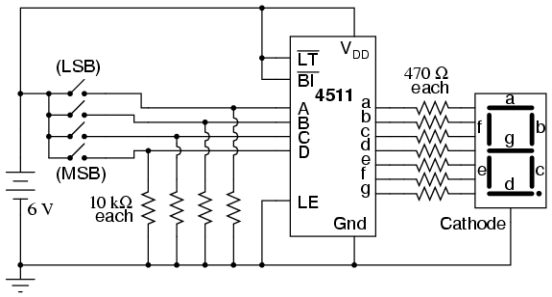
En dehors des codes purement arithmétiques, il existent de nombreux autres codes ayant un domaine d'application particulier. Nous n'en retiendrons que deux :
- le code BCD ;
- le code Gray.

3.5.1 Code BCD (binary coded decimal : décimal codé binaire)

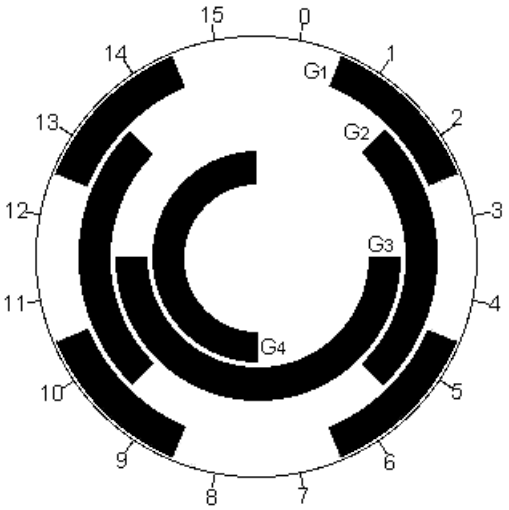
Chaque chiffre de la représentation décimale est codé en binaire naturel sur 4 bits.

Décimal	2	6	9
BCD	0010	0110	1001

Ce code est encore utilisé pour communiquer avec des dispositifs décimaux, comme certains afficheurs.



3.5.2 Code Gray



Le code **Gray** ou *binaire réfléchi* a été initialement créé pour résoudre les problèmes liés aux codeurs de position absolue.

Le passage d'une position à l'autre n'entraine alors le **changement que d'un seul bit**, évitant ainsi des codes intermédiaires fugitifs (dûs aux temps de réaction des capteurs, par exemple).

Les convertisseur le code binaire en code Gray et inversement sont très simples.

Si l'on désigne par b_n un bit quelconque en code binaire et par g_n le bit recherché en code Gray, on a alors : $g_n = b_n \oplus b_{n+1}$

Pour la conversion du code Gray en binaire, on a : $b_n = g_n \oplus b_{n+1}$

Méthode pour construire une table de code Gray à 2^n éléments

La méthode peut être décrite par l'algorithme suivant :

1. si $n = 2$, la table T_2 est composée des éléments 0 et 1 ;
2. si $n > 2$, la table T_n se construit à partir de la table T_{n-1}de taille $n - 1$ de la manière suivante :
   - on recopie la liste des $n - 1$ éléments de T_{n-1} ;
   - on ajoute la liste des $n - 1$ éléments de T_{n-1} en ordre inverse (en miroir) ;
   - on préfixe les $n - 1$ premiers éléments de 0 et les $n - 1$ éléments suivants de 1.

La méthode est illustrée ci-dessous pour les tables T_2, T_4 et T_8.

T_2	T_4	T_8
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

Notes

Notes (suite)


BIEN, CE COURS "SPECIAL MANAGERS" D'INITIATION A L'INFORMATIQUE SE TERMINE, AVEZ VOUS DES QUESTIONS ?
HUM!
NON!
NON!
TOUT EST CLAIR !
HEU... IL Y A JUSTE UN POINT QUI NE M'A PAS SEMBLÉ TRÈS EVIDENT !
VOYONS VOIR ÇA !
HE! HE!
C'ÉTAIT AU DEBUT DE LA SEMAINE !
ICI, ... ÇA DEVRAIT FAIRE 2 !?
?!
NON ?
"OU"
0+0=0
0+1=1
1+0=1
1+1=1
J'ALLAIS POSER LA MÊME QUESTION!
ÇA M'AVAIT FRAPPÉ AUSSI !
OUI, 1, C'EST PAS LOGIQUE!
IL A RAISON
FAUTE DE FRAPPE!
TSSSS!
ZEVAIR

Systèmes combinatoires

1. Introduction aux systèmes combinatoires

Comme tout système physique, les systèmes numériques peuvent être caractérisés par leur **temps de réponse** et leur **caractéristique d'entrée-sortie**. Les systèmes numériques ont besoin de **données binaires** en entrée et fournissent en sortie le même type d'informations.

Les fonctions liant l'entrée et la sortie d'un système binaire sont appelées des **fonctions logiques**. Elles sont régies par un ensemble de règles mathématiques appelées l'**algèbre de Boole**.

Dans le cas le plus simple, la notion de temps n'intervient pas et il existe une **relation directe entre l'entrée et la sortie**. On parle alors de systèmes **combinatoires**.

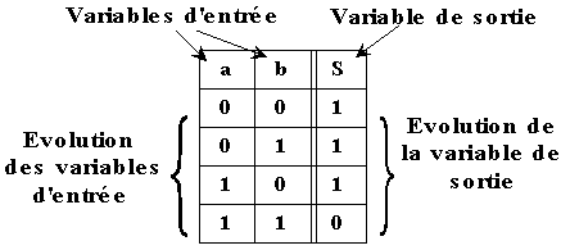
Les informations traitées par ces systèmes sont alors appelées des **variables logiques** ou **booléennes**. Elles ne peuvent avoir que **2 états** : "*vrai*" (ou '1' logique) ou "*faux*" (ou '0' logique). Ces niveaux logiques sont en pratique associés à des tensions.

2. Représentation des systèmes combinatoires

Il existe plusieurs façons de représenter un système combinatoire :
- par son **équation logique** ;
- par sa **table de vérité** ;
- par son **logigramme**.

L'**équation logique** permet de décrire **mathématiquement** la relation qu'il existe entre l'entrée du système et sa sortie. Chacune des sorties est une fonction des entrées. Cette représentation est basée sur l'**algèbre de Boole** (voir section suivante). Tout système combinatoire peut être réalisé à l'aide d'un petit nombre de fonctions logiques de base appelées **opérateurs logiques** ou **portes**.

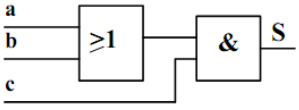
La **table de vérité** donne la liste des valeurs de sortie pour toutes les combinaisons possibles de l'entrée, classées selon l'ordre du code binaire naturel. Ainsi, la table de vérité d'une fonction de n variables a autant de ligne que d'états d'entrée, soit 2^n. Pour chacun de ces états, la sortie peut prendre la valeur "0" ou "1".



a	b	S
0	0	1
0	1	1
1	0	1
1	1	0

Le **logigramme** est un formalisme issu du **monde électronique**. Les expressions logiques sont traduites en un **câblage** reliant des symboles qui représentent les opérateurs.

Deux normes sont actuellement utilisées dans le monde, la plus ancienne est la norme ISO, toujours très utilisée dans le monde (en particulier par les logiciels de saisie de schéma). A cette norme, s'ajoute depuis 1984 la norme créée par la Commission Électrotechnique Internationale (IEC - norme IEEE 91-1984).



3. Algèbre de Boole

George BOOLE a défini, vers 1847, une algèbre qui s'applique à des fonctions logiques de variables booléennes.

Un ensemble E possède une structure d'algèbre de Boole si on a défini dans cet ensemble les éléments suivants :

- **Une relation d'équivalence** notée " = " ;
- **Deux lois de composition internes** notées " + " et " . " (addition et multiplication booléenne) ;
- **Une opération unaire** : loi qui associe à tout élément a de E son complément $\overline{a}$, cette loi est appelée **complémentation**.

Les lois énoncées ci-dessus s'écrivent :

+	**0**	**1**
0	0	1
1	1	1

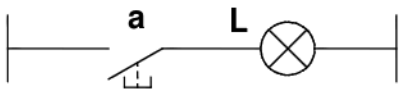
.	**0**	**1**
0	0	0
1	0	1

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

3.1. Opérateurs fondamentaux

3.1.1 L'opérateur OUI

Schéma électrique :



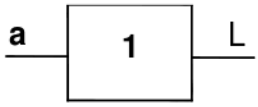
Equation :

L = a

Table de vérité :

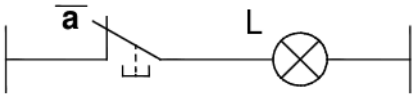
a	L
0	0
1	1

Symbole :



3.1.2 L'opérateur NON

Schéma électrique :



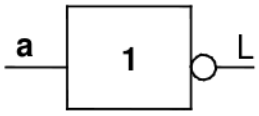
Equation :

$L = \overline{a}$

Table de vérité :

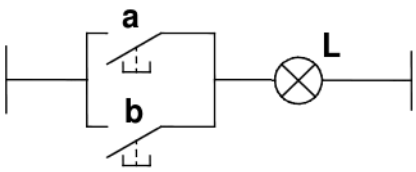
a	L
0	1
1	0

Symbole :



3.1.3 L'opérateur OU

Schéma électrique :



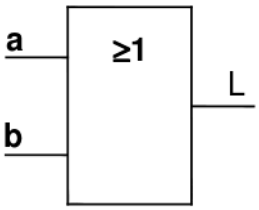
Equation :

L = a + b

Table de vérité :

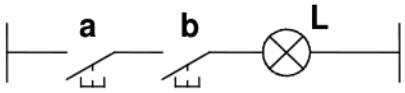
a	b	L
0	0	0
0	1	1
1	0	1
1	1	1

Symbole :



3.1.4 L'opérateur ET

Schéma électrique :



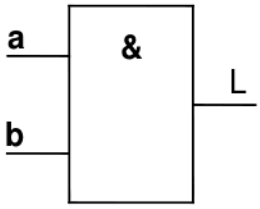
Equation :

L = a . b

Table de vérité :

a	b	L
0	0	0
0	1	0
1	0	0
1	1	1

Symbole :

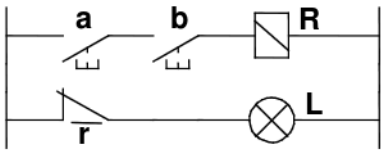


3.2. Opérateurs dérivés

A partir des opérateurs fondamentaux, on peut obtenir des opérateurs plus complexes. On parle alors d'**opérateurs dérivés**. Les 4 opérateurs suivants sont parmi les plus utilisés.

3.2.1 L'opérateur NON-ET (ou NAND)

Schéma électrique :



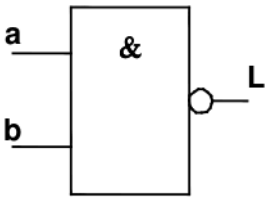
Equation :

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

Table de vérité :

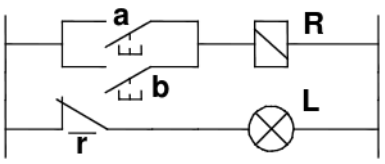
a	b	L
0	0	1
0	1	1
1	0	1
1	1	0

Symbole :



3.2.2 L'opérateur NON-OU (ou NOR)

Schéma électrique :



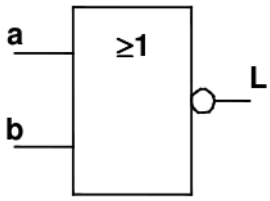
Equation :

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

Table de vérité :

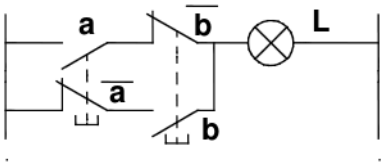
a	b	L
0	0	1
0	1	0
1	0	0
1	1	0

Symbole :



3.2.3 L'opérateur OU-EXCLUSIF (ou XOR)

Schéma électrique :



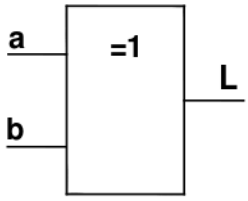
Equation :

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

Table de vérité :

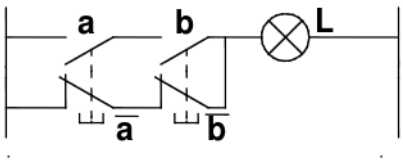
a	b	L
0	0	0
0	1	1
1	0	1
1	1	0

Symbole :



3.2.4 L'opérateur NON-OU-EXCLUSIF (ou XNOR)

Schéma électrique :



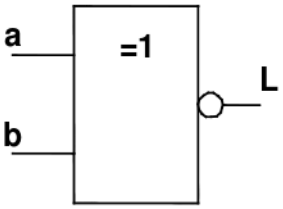
Equation :

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

Table de vérité :

a	b	L
0	0	1
0	1	0
1	0	0
1	1	1

Symbole :



3.3. Propriétés des opérations

3.3.1 Dualité des expressions

Chaque expression logique possède une **expression duale**.

L'expression duale est obtenue en échangeant les opérateurs et les valeurs logiques dans l'expression d'origine. Ainsi :

- l'opérateur + devient · ;
- l'opérateur . devient + ;
- la valeur logique "0" devient "1" ;
- la valeur logique "1" devient "0".

Par exemple, on a : $X + 0 = X$ qui devient : $X \cdot 1 = X$

3.3.2 Propriétés des opérations

Les colonnes gauche et droite représentent les formes duales.

Commutativité	
$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
Associativité	
$(X + Y) + Z = X + (Y + Z) = X + Y + Z$	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$
Distributivité	
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
Eléments neutres	
$X + 0 = X$	$X \cdot 1 = X$
$X + 1 = 1$	$X \cdot 0 = 0$
Complémentarité	
$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
Théorème d'idempotence	
$X + X = X$	$X \cdot X = X$

3.3.3 Théorèmes et axiomes de l'algèbre de Boole

Les colonnes gauche et droite représentent les formes duales.

Théorème d'involution
$\overline{(\overline{X})} = X$
Théorème de DeMorgan
$\overline{(X + Y + Z + ...)} = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdot ...$ $\quad \overline{(X \cdot Y \cdot Z \cdot ...)} = \overline{X} + \overline{Y} + \overline{Z} + ...$
$\overline{f(X_1, X_2, X_3, ..., X_n, 0, 1, +, \cdot)} = f(\overline{X_1}, \overline{X_2}, \overline{X_3}, ..., \overline{X_n}, 1, 0, \cdot, +)$
Théorème de multiplication et de factorisation
$(X + Y) \cdot (\overline{X} + Z) = X \cdot Z + \overline{X} \cdot Y$ $\quad X \cdot Y + X \cdot \overline{Z} = (X + Z) \cdot (\overline{X} + Y)$
Théorème du consensus
$X \cdot Y + Y \cdot Z + \overline{X} \cdot Z = X \cdot Y + \overline{X} \cdot Z$ $\quad (X + Y) \cdot (Y + Z) \cdot (\overline{X} + Z) = (X + Y) \cdot (\overline{X} + Z)$

4. Synthèse de circuits combinatoires

La **synthèse** de fonctions combinatoires consiste, à partir d'une table de vérité ou d'une expression booléenne, à spécifier les **opérateurs matériels** permettant l'implémentation de la table ou de l'expression correspondante dans un système réel. En effet, chaque **opérateur élémentaire** possède sa solution technologique sous forme de **circuit intégré** (circuits CMOS : 4001 / porte XOR, 4081 / porte ET, ...).

Dans la pratique, les systèmes combinatoires peuvent rapidement devenir complexes à mettre en oeuvre. L'équation complète régissant les sorties en fonction des entrées n'est pas nécessairement la **forme minimale** et donc pas la forme la plus économique à réaliser. Avant d'aborder l'implémentation du système sur une carte, il est donc intéressant de **simplifier l'équation** afin d'obtenir le nombre minimal d'opérateurs logiques.

Pour cela, il existe au moins trois méthodes différentes :
- une **méthode analytique**, se basant sur les théorèmes de l'albèbre de Boole ;
- une **méthode graphique**, se basant sur l'utilisation de tableau de Karnaugh ;
- une **description comportementale** [2], à l'aide de langage de description matérielle de haut niveau (type VHDL ou Verilog).

4.1. Méthode analytique de simplification d'équations

Le but de la simplication d'équations booléennes est de **réduire** au maximum **le nombre d'opérateurs logiques** qui interviennent dans la relation entrées-sorties du système.

Pour cela, on peut utiliser tous les **outils algébriques** qui sont mis à notre disposition dans l'algèbre de Boole (voir section précédente) ainsi que les quelques règles de simplification suivantes.

Théorèmes de simplification	
$X \cdot Y + X \cdot \overline{Y} = X$	$(X + Y) \cdot (X + \overline{Y}) = X$
$X + X \cdot Y = X$	$X \cdot (X + Y) = X$
$(X + \overline{Y}) \cdot Y = X \cdot Y$	$(X \cdot \overline{Y}) + Y = X + Y$

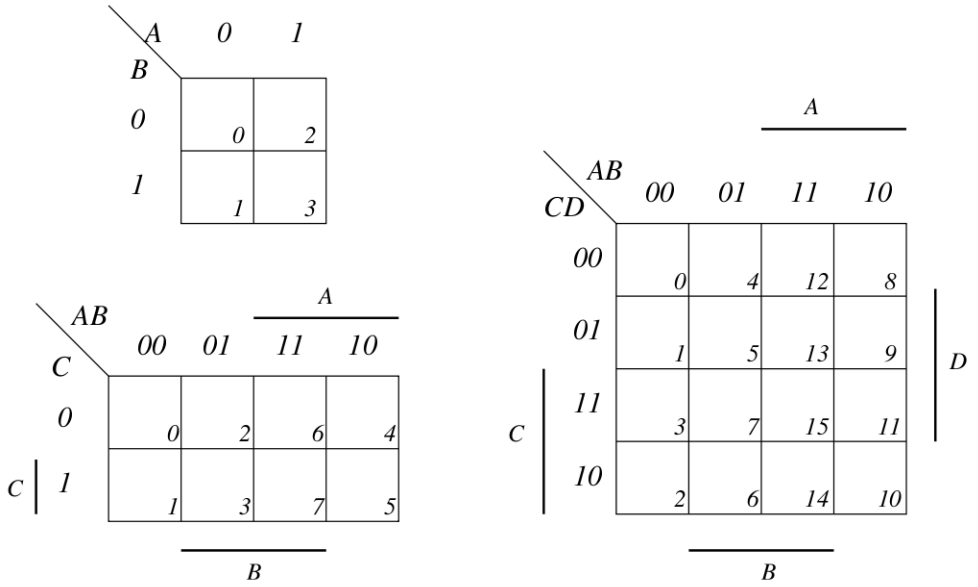
4.2. Méthode graphique de simplification d'équations (Karnaugh)

Les simplifications précédentes peuvent être réalisées graphiquement à l'aide d'un **tableau de Karnaugh**. Cette méthode se fonde sur une manière de représenter la table de vérité qui fait apparaître des **symétries sur les variables**, et en particulier sur le **théorème d'unification** : $A \cdot (B + \overline{B}) = A$

Pour rendre plus visibles les cases adjacentes (et donc les symétries), la table de vérité linéaire à numérotation binaire naturelle est remplacée par une **table rectangulaire** dont la numérotation des lignes et des colonnes suit la logique du **code Gray** (une seule variation d'une position à la position suivante).

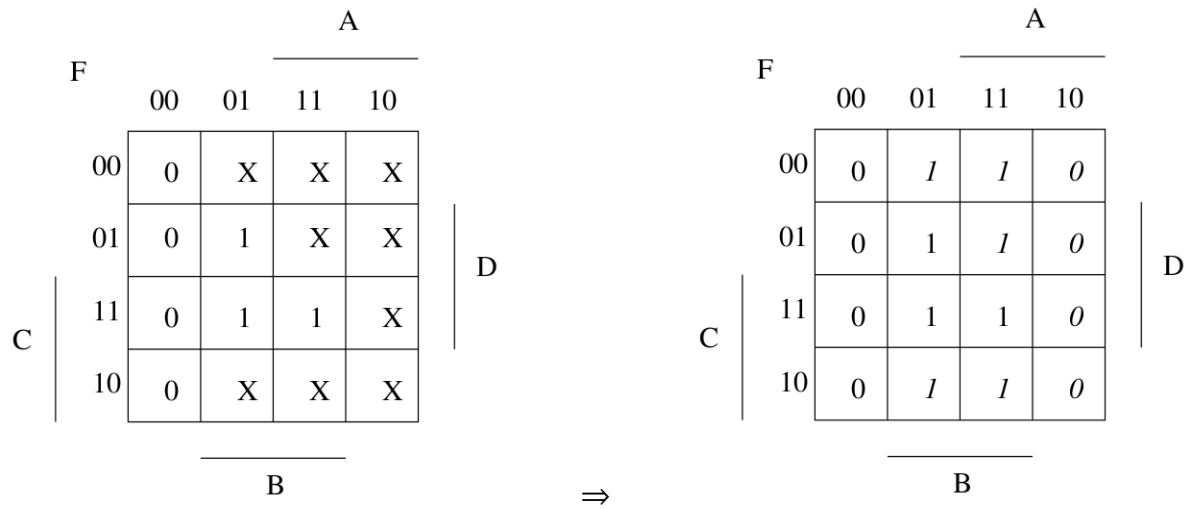
La méthode du diagramme de Karnaugh est efficace pour les expressions booléennes ayant au plus 4 entrées. Au delà, la représentation graphique devient complexe, il est difficile de mettre en évidence les symétries, et la méthode devient inutilisable. Les variables sont affectées (par couples le cas échéant) aux colonnes et lignes du tableau. La fonction est reportée dans les cases du tableau. On peut aussi reporter dans les cases les numéros d'indexation.

2. Cette dernière méthode sera étudiée plus en détail dans une autre partie du cours, ainsi qu'en travaux pratiques.



Les règles pour la simplification des fonctions booléennes avec le tableau de Karnaugh sont les suivantes :
- tous les termes pour lesquels la fonction est à 1 devront être pris au moins une fois dans un **regroupement**, ou seuls si aucun regroupement n'est possible ;
- les regroupements doivent être de **taille maximale**, de manière à éliminer le plus grand nombre possible de variables dans les termes de l'expression ;
- les regroupements doivent contenir un nombre égal à une **puissance de deux** d'éléments ;
- les recouvrements entre regroupements sont possibles ;
- une case d'un bord est aussi **adjacente** à celle correspondante du bord opposé ;
- un regroupement de 2 cases permet l'élimination d'une variable, un regroupement de 4 cases l'élimination de deux variables, etc...

Les problèmes réels conduisent souvent à des spécifications incomplètes des fonctions logiques. Des valeurs indéterminées (notées 'X') se présentent alors dans le tableau. On peut adopter la valeur qui conduit à la meilleure simplification.

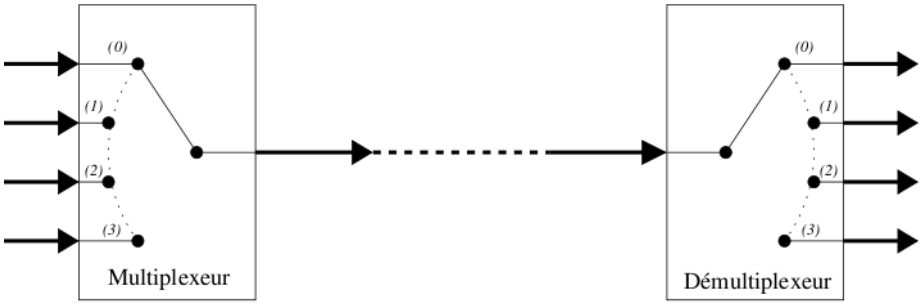


5. Fonctions combinatoires standard

Il existe des fonctions combinatoires plus complexes que les opérateurs logiques vus jusqu'à maintenant qui sont régulièrement utilisés dans les systèmes numériques.

5.1. Multiplexeurs / Démultiplexeurs

Les fonctions logiques **multiplexeur** et **démultiplexeur** sont en relation étroite avec les problèmes liés aux **transmissions d'informations multiples** par un canal de transport unique. Ce problème s'est posé dès les premières transmissions téléphoniques. Comment relier plusieurs sources à plusieurs destinataires via un seul (et coûteux) câble ?

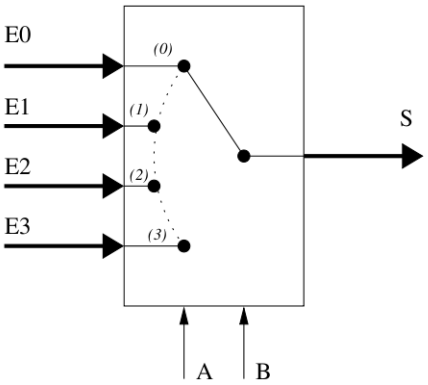


Bien entendu, il n'était pas question de transmettre les signaux des sources *simultanément* mais *successivement*. Les centraux téléphoniques sont une image de cette fonction.

Les dénominations habituelles sont :
- un multiplexeur "1 parmi N" (N entrées et 1 sortie) ;
- un démultiplexeur à N voies (1 entrée et N sorties).

5.1.1 Réalisation d'un multiplexeur pour 4 signaux binaires

La commande du sélecteur de voie (1 parmi N) est réalisée en utilisant le codage binaire du numéro de la voie choisie (de 0 à N-1). Ce code est appliqué au moyen de 2 variables logiques de sélection A et B. Le schéma et la table de vérité correspondants sont donnés ci-dessous :



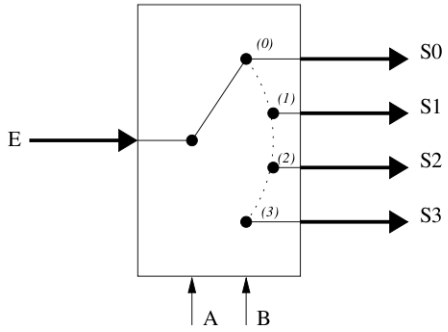
N°	A	B	S
0	0	0	E0
1	0	1	E1
2	1	0	E2
3	1	1	E3

L'expression logique de la fonction S se déduit aisément :

$$S = \overline{A}.\overline{B}.E_0 + \overline{A}.B.E_1 + A.\overline{B}.E_2 + A.B.E_3$$

5.1.2 Réalisation d'un démultiplexeur pour 4 signaux binaires

Un raisonnement similaire conduit au schéma et aux tables de vérité du démultiplexeur à 4 voies représenté ci-dessous.

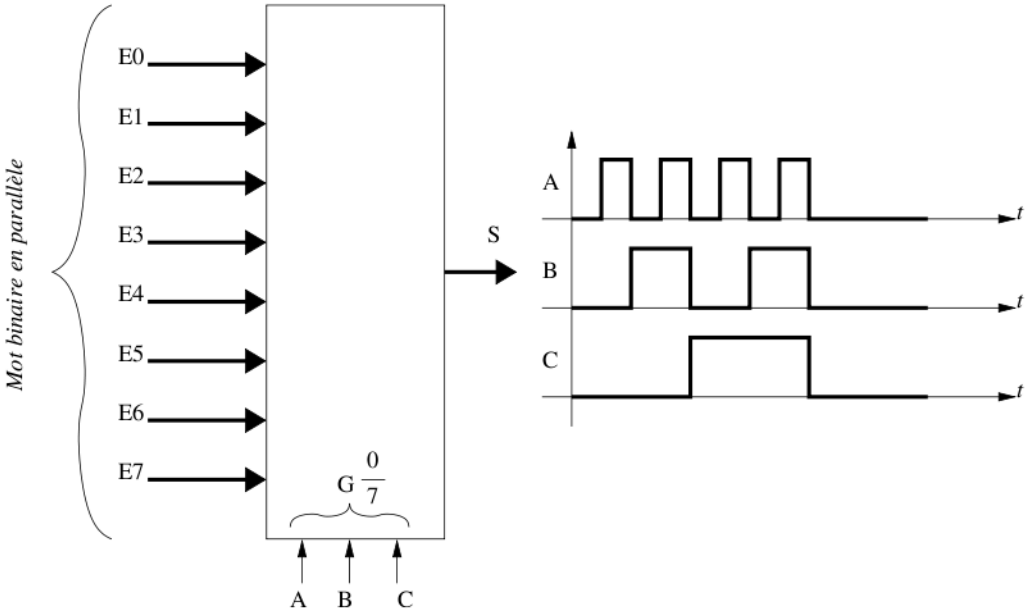


N°	A	B	S0	S1	S2	S3
0	0	0	E	0	0	0
1	0	1	0	E	0	0
2	1	0	0	0	E	0
3	1	1	0	0	0	E

Les fonctions logiques S_0, S_1, S_2, S_3 se déduisent aisément :$S_0 = \overline{A}.\overline{B}.E$, $S_1 = \overline{A}.B.E$, $S_2 = A.\overline{B}.E$, $S_3 = A.B.E$.

5.1.3 Convertisseurs parallèle-série

Associé à un compteur (autre système logique qui sera étudié par la suite dans ce module) qui génère les signaux A, B et C de la façon décrite ci-dessous sur les entrées de sélection, le multiplexeur permet de **convertir une donnée parallèle vers une liaison série**.



5.2. Codeurs et décodeurs

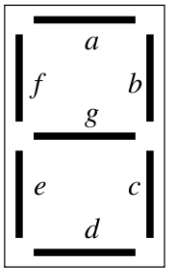
Ce sont des dispositifs qui assurent la traduction d'une représentation de l'information vers une autre. La dénomination varie selon les usages.

Codeurs

– Codeur binaire : possédant N entrées numérotées de 0 à $N-1$, il délivre sur ses sorties le code binaire du numéro de l'entrée activée. S'il est possible que plusieurs entrées soient simultanément activées, ce sera le code de l'entrée de plus fort numéro qui sera présent (codeur de priorité).

Décodeurs

– Décodeur décimal : cette fonction transforme le code binaire naturel d'un nombre compris entre 0 et 9 en une indication décimale sous la forme de l'activation d'une des 10 lignes de sortie : le code binaire 0000 active la sortie S_0, le code 0001 la sortie S_1, etc.. jusqu'au code 1001 qui active la sortie S_9.
– Décodeur "7 segments" : il fonctionne à la manière du précédent, mais au lieu de fournir un code de sortie "un parmi dix", il produit une combinaison de signaux aptes à allumer les segments d'un afficheur à 7 segments pour former le chiffre correspondant.



d3	d2	d1	d0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

Transcodeurs

– Transcodeur Gray vers binaire : à partir d'un code de Gray à N bits, il fournit le code binaire correspondant.

5.3. Fonctions arithmétiques

La plupart des **fonctions arithmétiques** sont indispensables dans les circuits numériques : addition de variables, multiplication de signaux... Les **microcontroleurs**[3], par exemple, possèdent une **unité de calculs arithmétiques et logiques** précablée.

Tous ces calculs arithmétiques peuvent se faire à l'aide de circuits de la logique combinatoire.

5.3.1 Comparateur de 2 nombres entiers positifs

Soient 2 nombres entiers A et B codés sur n bits, le problème posé est de disposer de signaux logiques $S_=$, $S_<$ et $S_>$ indiquant respectivement si $A = B$, $A < B$ et $A > B$.

Sur des nombres codés sur 2 bits par exemple, on obtient la table de vérité suivante :

A		B		$S_=$	$S_<$	$S_>$
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0

A		B		$S_=$	$S_<$	$S_>$
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

3. Ces composants seront étudiés plus tard dans l'année.

5.3.2 Additionneur à un bit (demi-additionneur)

La somme de deux nombres codés en binaire naturel sur un bit nécessite une représentation sur 2 bits (un bit de somme et un bit de retenue). Soient *a* et *b* les 2 nombres, *s* le bit somme et *co* le bit de retenue. Leurs tables de vérité respectives sont indiquées ci-dessous :

a	*b*	*s*	*co*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ce circuit est souvent appelé **demi-additionneur** : il ne peut être utilisée dans une addition à plus d'un bit, car il ne dispose pas d'une entrée permettant de prendre en compte une retenue provenant d'une addition de rang inférieur.

5.3.3 Additionneur à plusieurs bits, additionneur complet

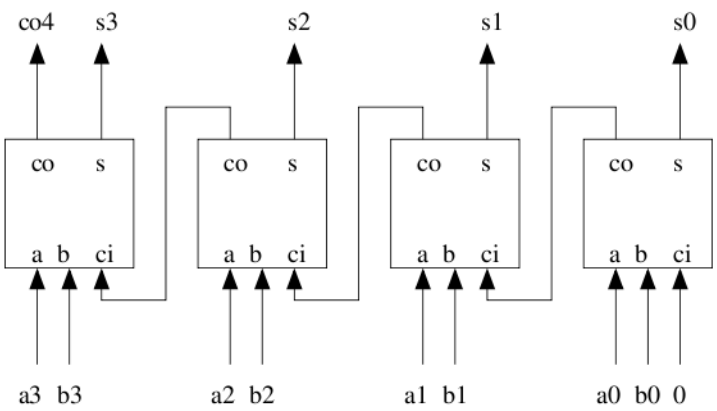
L'addition à plusieurs bits se réalise à la main avec le même algorithme que l'addition décimale :

retenues	1	1	1	1	
		1	1	0	1
+		1	0	1	1
	1	1	0	0	0

On s'aperçoit que l'addition doit traiter, en général, 3 opérandes : *a*, *b* et la retenue *ci*. Il doit toujours produire 2 résultats *s* et *co*. On obtient alors la table de vérité de l'additionneur complet :

a	*b*	*ci*	*s*	*co*
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

Pour réaliser un **additionneur à plusieurs bits**, il suffit alors de mettre en cascade plusieurs **additionneurs complets**.



5.3.4 Multiplication de 2 nombres entiers positifs

L'algorithme de multiplication n'est pas lié à la base de numération, nous pouvons donc procéder de la même manière qu'en base 10 : additions répétées et décalages à gauche.

La multiplication de 2 nombres à plusieurs bits s'effectue en calculant les produits partiels décalés selon le rang du bit du multiplicateur, puis en ajoutant ces produits partiels.

				1	0	1	1
			×	1	1	0	1
				1	0	1	1
			0	0	0	0	
		1	0	1	1		
	1	0	1	1			
1	0	0	0	1	1	1	1

Notes

Notes (suite)

Annexe 1 : Formes canoniques des expressions logiques

Afin de comparer des expressions booléennes exprimées sous forme algébrique, il est utile de disposer d'une forme standard qui représente la fonction. Une telle forme standard est appelée *forme canonique*. Il existe deux formes canoniques couramment employées : *somme de produits* et *produit de sommes*.

Pour constituer de telles formes, on part de la table de vérité d'une fonction logique. Nous considèrerons la fonction F ci-dessous, ainsi que sa forme complémentée $\overline{F}$.

A	B	C	F	$\overline{F}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

8.1. Somme de produits

Cette forme est aussi appelée *développement en «minterms»* ou *forme disjonctive normale*. Pour obtenir cette forme, on considère chaque «1» de la table de vérité de F. On peut alors lire la table de vérité ainsi :

F=1 si (A=0 et B=1 et C=0) ou (A=1 et B=0 et C=0) ou (A=1 et B=0 et C=1) ou (A=1 et B=1 et C=0) ou (A=1 et B=1 et C=1).

Cette équivalence peut se formuler autrement :

$$F = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

et de même :

$$\overline{F} = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}$$

Ce sont les expressions en *somme de produits* de F et de $\overline{F}$.

Mais, en utilisant les propriétés de la numération binaire pour identifier les lignes, on peut s'abstenir de développer complètement les produits, mais on peut indiquer simplement le nombre binaire constitué par la combinaison de A, B et C pris dans cet ordre précédé de la lettre m faisant référence à *minterm*. Par exemple, pour F, les minterms seront m_3, m_4, m_5, m_6, m_7. Nous écrirons alors :

$$F(A,B,C) = m_3 + m_4 + m_5 + m_6 + m_7 = \sum m(3,4,5,6,7)$$

Nous aurons aussi :

$$\overline{F}(A,B,C) = \sum m(0,1,2) = m_0 + m_1 + m_2$$

Il est à remarquer que le développement en minterms ne garantit pas que la forme obtenue est minimale. On peut d'ailleurs le vérifier car F peut se mettre sous la forme : $F = B \cdot C + A$.

8.2. Produit de sommes

La forme en produit de sommes est parfois appelée *forme conjonctive normale* ou bien *développement en maxterms.*

En appliquant le théorème de DeMorgan à l'expression en somme de produits de $\overline{F}$ rappelée ci-dessous :

$$\overline{F} = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}$$

On obtient :

$$F = (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C)$$

En utilisant cette fois la numération binaire complémentée pour identifier les lignes correspondantes de la table de vérité, et le symbole M faisant référence à *Maxterm* :

$$F(A, B, C) = \prod M(0, 1, 2) = M_0 \cdot M_1 \cdot M_2$$

8.3. Passage d'une forme canonique à une autre

Pour réaliser rapidement ce passage, il suffit de remarquer que les minterms et les Maxterms sont mutuellement exclusifs dans les développements : si un minterm m_i est présent dans le développement en minterms, le Maxterm de même indice M_i sera absent du développement en Maxterms. Donc, pour passer d'un développement à l'autre, on doit rechercher tous les termes absents du développement d'origine, les changer de nom, puis transformer les sommes en produits et vice versa.

Par exemple :
$F(A, B, C) = \prod M(0, 2, 3) = M_0 \cdot M_2 \cdot M_3 \Leftrightarrow F(A, B, C) = \sum m(1, 4, 5, 6, 7) = m_1 + m_4 + m_5 + m_6 + m_7$

8.4. Fonctions à spécification incomplète

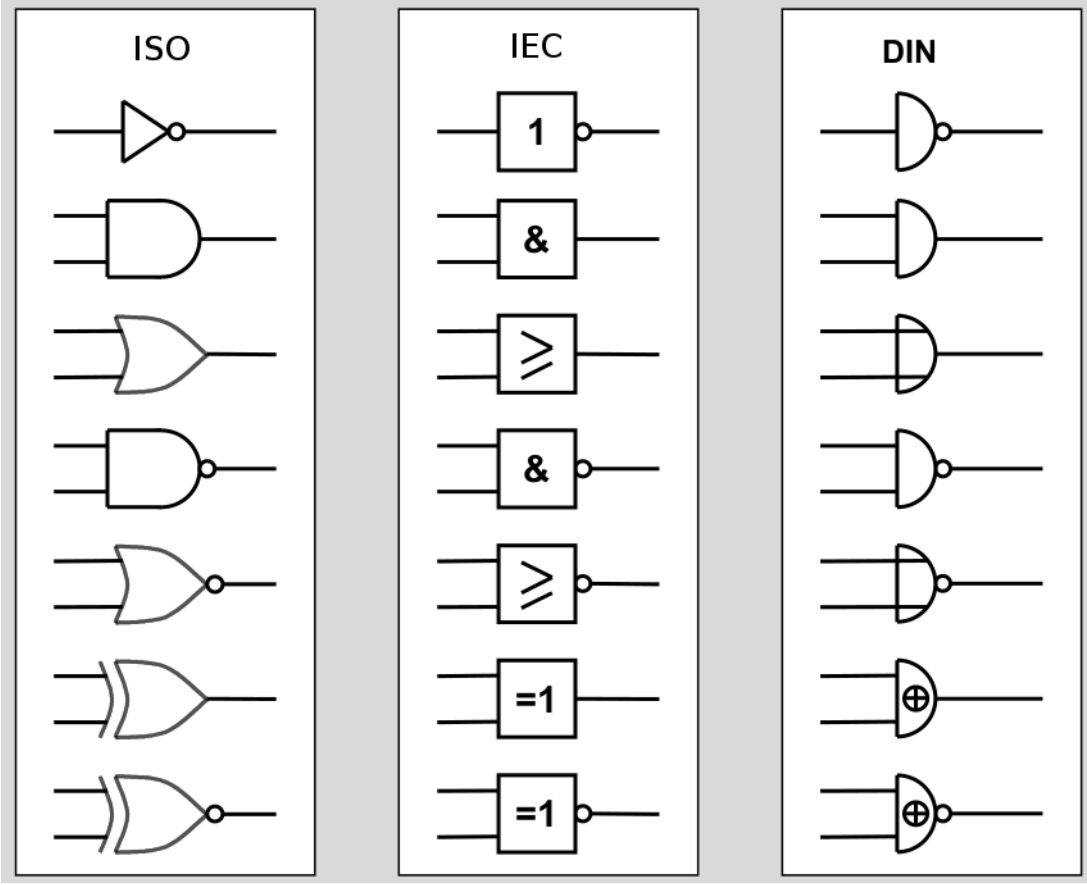
Il arrive fréquemment dans les applications réelles que pour certaines combinaisons de variables (ou assertions), la fonction logique étudiée ne soit pas spécifiée (ni vraie, ni fausse). Dans ce cas, le concepteur a généralement le choix pour fixer le niveau logique à une valeur arbitraire, très souvent en vue de simplifier sa conception. Mais en tout état de cause, on doit compléter les expressions en minterms ou en Maxterms par des *indifférents* (don't care) notés d_j dans le développement en minterms et D_j dans le développement en Maxterms. A noter que contrairement aux min- et Max- terms, les indifférents sont communs aux deux développements.

Par exemple :

$$F(A, B, C) = \prod M(0, 2, 3) D(4, 7) = M_0 \cdot M_2 \cdot M_3 \cdot D_4 \cdot D_7$$

$$\Leftrightarrow F(A, B, C) = \sum m(1, 5, 6) + d(4, 7) = m_1 + m_5 + m_6 + d_7$$

Annexe 2 : Symboles normalisés des Portes Logiques

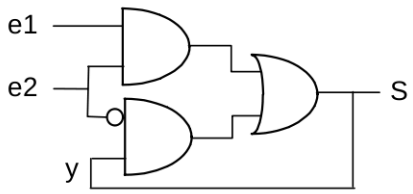


Systèmes séquentiels

1. Introduction aux systèmes séquentiels

Dans les **systèmes combinatoires**, les valeurs des sorties à un instant donné sont directement imposées par celles des entrées.

Mais que se passe-t-il si on reboucle l'une des sorties d'un tel système sur l'une de ses entrées ?



e1	e2	S
0	0	y
0	1	0
1	0	y
1	1	1

De manière générale, les systèmes numériques qui font apparaitre des boucles de rétroaction permettent de mémoriser des informations relatives aux stimuli antérieurs appliqués sur le circuit. La sortie d'un tel circuit, en plus des variables d'entrées et de sorties, est aussi fonction de **variables internes** (y dans l'exemple précédent), appelées **variables d'état**.

Ces systèmes sont appelés des **systèmes séquentiels**. Ils sont caractérisés par le fait que pour chaque combinaison de variables d'entrée (chaque état d'entrée), les sorties peuvent prendre plusieurs valeurs possibles (plusieurs états de sortie). La valeur présente en sortie dépend également de l'**état précédent du système**. Il est alors nécéssaire de pouvoir stocker cet état et donc d'introduire la fonction de **mémorisation** d'une grandeur binaire.

1.1. Notion d'état

Les valeurs des variables internes d'un système évoluent au fil du temps, en fonction des changements sur les entrées et des valeurs des sorties. A chaque combinaison de valeurs des variables du système correspond une configuration de celui-ci qu'on appelle l'**état du système**.

On identifie un état du système par un ensemble de variables de ce système appelées **variables d'état**. Elles correspondent au plus petit ensemble de variables indépendantes qui permettent de coder l'état du système. Lorsqu'il existe p variables d'état, il existe 2^p états possibles.

Remarque : Le nombre de variables d'état n'est pas unique : on peut toujours utiliser plus de variables d'état que le strict nécessaire. On peut par exemple faire correspondre à chaque état une variable d'état qui passe '1' lorsque l'état correspondant est atteint : codage *un parmi n*.

1.2. Systèmes synchrones ou asynchrones

Un système séquentiel peut être :

- **synchrone** : son évoluation (et en particulier celle de ses sorties) est controlable de l'extérieur par un signal appelé **horloge** ;
- **asynchrone** : son évolution n'est pas controlable de l'extérieur.

Les systèmes séquentiels **asynchrones** sont plus rapides, mais plus difficiles à mettre au point dans les applications réelles, puisque dès lors qu'un changement intervient sur son entrée, il peut provoquer un changement sur ses sorties instantanément.

Les systèmes séquentiels **synchrones** sont moins efficaces, mais plus fiables et plus préictibles, l'évolution de leur sortie étant directement controlée par une horloge.

Nous nous intéresserons dans la suite de ce cours uniquement à la synthèse des systèmes synchrones (sauf compteurs asynchrones).

2. Représentation des systèmes séquentiels

La succession des divers états pris par le système au cours de son fonctionnement constituent ce qu'on appelle une **séquence** (d'où le terme de systèmes séquentiels).

Chaque état d'une machine à état peut être représenté par l'ensemble des valeurs de ses variables d'état : à un ensemble de valeurs donné correspond **un état unique**.

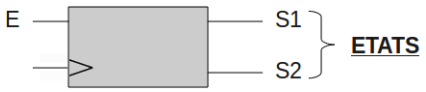
Il est courant d'**identifier un état donné** par un symbole :
- une lettre ou un groupe de lettre ayant une signification dans le système considéré ;
- un nombre servant de numéro à l'état considéré.

L'évolution d'une machine à état est guidé par les valeurs des variables d'entrée. La **transition** d'un état à l'autre dépend de ces valeurs.

Il existe plusieurs façons de représenter ces séquences et ces changements d'état. On peut utiliser :
- une **équation logique** ;
- un **chronogramme** ;
- une **table de transition** ;
- un **diagramme d'état** ;
- une **structure** logique.

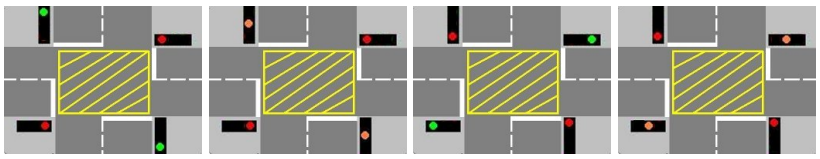
Il existe également d'autres représentations possibles de ces systèmes tels que les **GRAFCET** ou les **réseaux de Pétri**. Ces représentations sont souvent associées à un type particulier de machine qu'on appelle des **automates programmables industriels** (API).

Pour pouvoir illustrer les différentes représentation possible d'un système séquentiel, nous allons prendre l'exemple de la gestion d'un *carrefour à 2 voies de circulation*.



Le système est commandé par une entrée E. La machine change d'état à chaque changement de valeur sur cette entrée E.

4 états sont nécessaires pour représenter ce système, il faudra au minimum deux variables d'état pour les coder. Ces variables d'état seront appelés $S1$ et $S2$

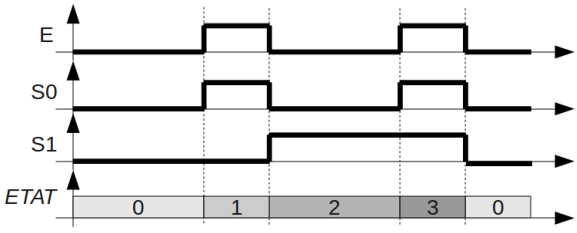


ETAT 0 ETAT 1 ETAT 2 ETAT 3

2.1. Chronogramme

Les **chronogrammes** sont très répandus dans le monde de l'électronique pour pouvoir représenter l'**évolution temporelle** d'un signal. C'est un modèle graphique qui représente l'évolution au cours du temps de toutes les entrées et sorties du système.

On ne représente pas toujours le signal d'horloge. Mais sur un système synchrone, un changement en sortie ne peut avoir lieu que lors d'un **front actif** sur l'entrée d'horloge.



Cette représentation permet de définir un certain nombre d'états du système. Dès que l'on augmente le nombre d'entrées/sorties, il existe un risque d'oublier certains de ces états. Ce mode de représentation n'est pas synthétique et doit être réservé à la représentation d'un **exemple concert de fonctionnement** du système et non à l'intégralité du fonctionnement.

2.2. Table de transition

Une **table de transition** permet de représenter dans un tableau la **transition** qu'il existe d'un état à l'autre en fonction des entrées du système. C'est une représentation équivalente à la table de vérité pour les systèmes combinatoires avec d'un coté les entrées et les états actuels du système et de l'autre les états suivants du système.

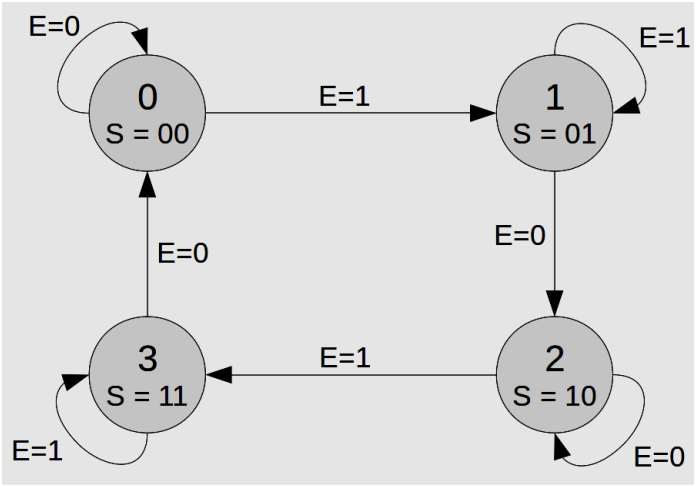
Etats actuels $S_1^- S_0^-$	E E	Etats suivants $S_1^+ S_0^+$
0 0	0	0 0
0 0	1	0 1
0 1	0	1 0
0 1	1	0 1
1 0	0	1 0
1 0	1	1 1
1 1	0	0 0
1 1	1	1 1

C'est à partir de cette représentation qu'il est le plus simple de concevoir la structure d'un système numérique séquentiel. Cette étude, appelée **synthèse**, sera étudiée dans une des sections suivantes de ce cours.

2.3. Diagramme d'état

Un **diagramme d'état** permet de visualiser l'ensemble des états du système et des transitions existantes de manière graphique. Chaque état est représenté dans un **cercle** portant le nom de l'état. Dans ce cercle est également indiquée la **valeur des sorties**.

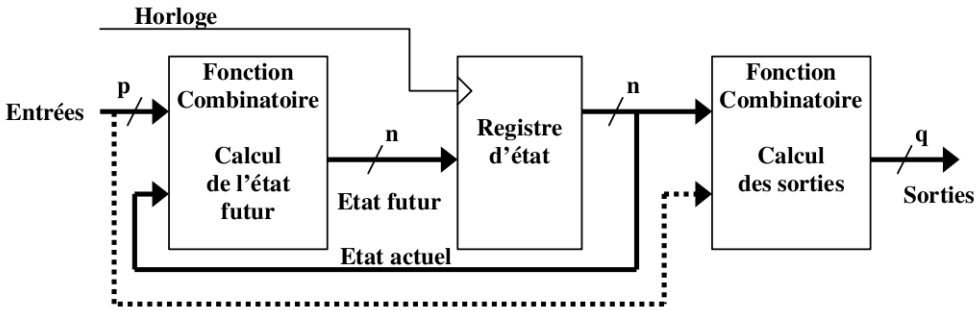
Entre différents états, il peut exister des **transitions**. Elles sont matérialisés par des **flèches**, qui indiquent le sens de la transition ainsi que la condition de franchissement de celle-ci. Ces franchissements ne peuvent avoir lieu, dans un système synchrone, que sur **front actif de l'horloge**.



2.4. Structure d'une machine à état

Enfin, il est possible de représenter un système séquentiel par sa **structure complète**, tel un logigramme. Un tel système peut être schématisé par :

- un **élément de mémorisation**, retenant la valeur des variables d'état pour l'état actuel ;
- un **élément de calcul**, pour déterminer la valeur des variables d'état pour l'état futur ;
- optionnellement, un élément de calcul supplémentaire qui élabore les valeurs de sortie à partir des variables d'état.

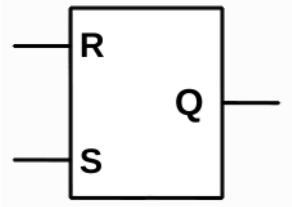


3. Bascules : composants élémentaires de la logique séquentielle

Dans la structure précédente, nous avons déjà vu comment réaliser des fonctions de calcul à l'aide de fonctions combinatoires. Nous allons à présent nous intéresser aux composants de base permettant la réalisation de la **fonction mémoire**. Ces composants s'appellent des **bascules**.

3.1. Bascule RS

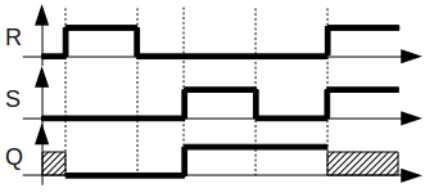
La bascule la plus simple est la **bascule RS**. C'est une mémoire possèdant deux entrées R et S et une sortie Q. Le S signifie **Set** (ou mise à un) et R signifie **Reset** (ou mise à zéro).

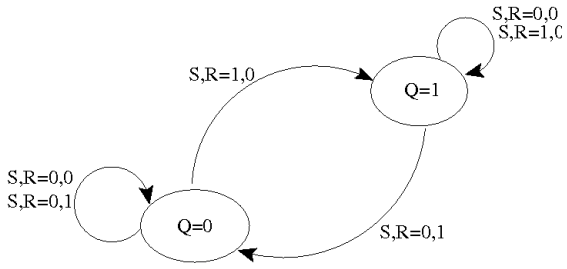


Lorsque l'entrée S vaut '1', la sortie Q passe à '1' (mise à un).
Lorsque l'entrée R vaut '1', la sortie Q passe à '0' (mise à zéro).
Lorsqu'aucune des entrées n'est à '1', alors la bascule conserve la valeur précédente. C'est l'**état mémoire**.

Son équation est la suivante :

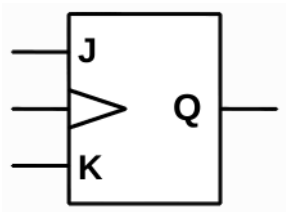
$$Q^+ = S + R \cdot Q^-$$





Malgré le fait qu'elle soit **asynchrone** (pas de signal d'horloge) et qu'il existe un état interdit (R=1 et S=1), cette bascule est encore utilisée dans un certain nombre de procédés tels que les **systèmes anti-rebond**.

3.2. Bascule JK



La **bascule JK** est considérée comme la version synchrone de la bascules RS. C'est une mémoire possèdant deux entrées J et K et une sortie Q. Etant synchrone, elle possède également une entrée spécifique d'horloge CLK. Le J signifie **Jump** (ou mise à un) et K signifie **Knock** (ou mise à zéro).

Son équation est la suivante :

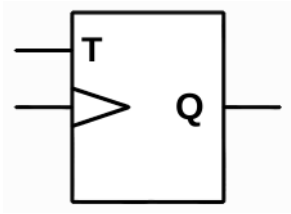
$$Q^+ = Q^- \cdot \overline{K} + \overline{Q^-} \cdot J$$

Cette bascule permet 4 fonctions différentes, résumées dans la table de transition suivante :

J	K	Q^+	Fonction
0	0	Q^-	mémoire
0	1	0	reset (knock)
1	0	1	set (jump)
1	1	$\overline{Q^-}$	toggle / basculement

Remarque : le signal d'horloge n'intervient pas dans l'écriture de l'équation, ni même dans la table de transition, mais **la sortie ne peut évoluer que sur un front actif** (montant ou descendant, selon la bascule) de ce signal d'horloge.

3.3. Bascule T



Cette bascule élémentaire est essentiellement utilisée dans la réalisation de compteurs (voir section 4.1). Le T signifie *toggle* (ou basculement). Elle possède une entrée d'horloge CLK, une entrée T et une sortie Q. Elle est **synchrone**.

Elle change d'état à chaque front actif d'une horloge (entrée CLK) lorsque $T = 1$ et conserve son état lorsque $T = 0$.

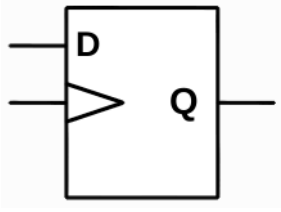
Son équation est la suivante :

$$Q^+ = T \cdot \overline{Q^-} + \overline{T} \cdot Q^-$$

Son fonctionnement peut aussi être représenté par la table de transition suivante :

T	Q^+	Fonction
0	Q^-	mémoire
1	$\overline{Q^-}$	toggle / basculement

3.4. Bascule D



La **bascule D** est probablement la plus utilisée de toutes les bascules. On la retrouve dans un grand nombre de systèmes séquentiels, entre autre dans les circuits logiques complexes de type CPLD [4] ou FPGA [5]. Elle possède une entrée d'horloge CLK, une entrée de donnée D et une sortie Q.

Cette bascule recopie l'entrée de donnée D sur sa sortie Q à chaque front actif de l'horloge.

Son équation est la suivante :

$$Q^+ = D$$

4. Complex Programmable Logic Device
5. Field Programmable Gate Array

4. Fonctions séquentielles standard

Il existe des fonctions séquentielles plus complexes que les bascules, qui sont régulièrement utilisés dans les systèmes numériques.

Nous allons étudier plus en détail par la suite :

– les compteurs / décompteurs
– les registres

4.1. Compteurs / Décompteurs / Diviseurs de fréquence

Un **compteur** est une association de n bascules permettant de décrire, au rythme d'une horloge, une séquence déterminée qui peut avoir au maximum 2^n combinaisons différentes. Les combinaisons apparaissent toujours dans le même ordre.

Lorsque la succession des états correspondra à un ordre croissant, on utilisera le terme de **compteur**, et dans le cas contraire, le terme de **décompteur**.

On peut également parler de **modulo** lorsqu'il est question de compteur. Le **modulo** est le **nombre d'états différents** que peut prendre un compteur.

Un compteur modulo N, par exemple, démarre de la valeur 0 et compte dans l'ordre binaire naturel jusqu'à $N - 1$.

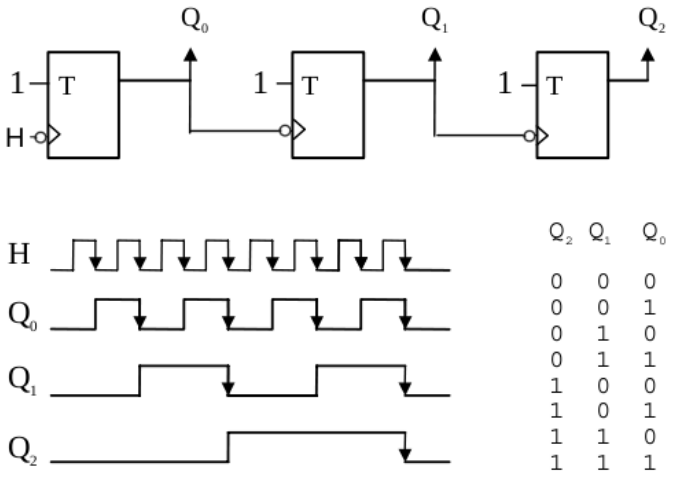
On appliquera souvent un qualificatif pour caractériser un compteur, selon le codage des états et le nombre de ceux-ci :

– compteur **binaire** si les états correspondent à un codage binaire naturel des variables d'état ;
– compteur **décimal** ou BCD dans le cas où il s'agit d'un codage BCD des variables d'état ;
– d'autres codages (Johnson, sexagésimal, bi-quinaire...).

4.1.1 Compteurs asynchrones

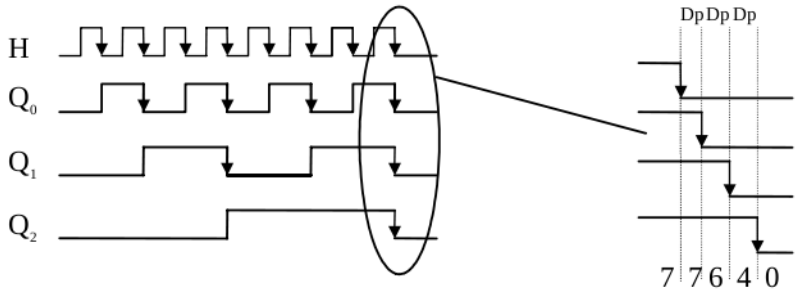
La façon la plus simple de réaliser un compteur est de remarquer d'une bascule T, dont l'entrée est à '1' en permanence, se comporte comme un **diviseur de fréquence par 2** de l'horloge d'entrée.

Ainsi la mise en cascade de plusieurs bascules de ce type permet de propager l'état futur d'une bascule à l'autre.



Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

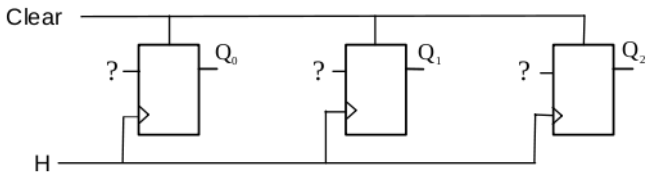
Ce type de structure n'est cependant pas recommandé car il peut être source de nombreux ennuis. En effet, le fait de mettre en cascade ces bascules entraine également un **cumul des temps de réaction** (ou propagation) de chacune d'entre elles. Ainsi on peut voir apparaitre à certains moments de la séquence des **états non désirés**.



4.1.2 Compteurs synchrones

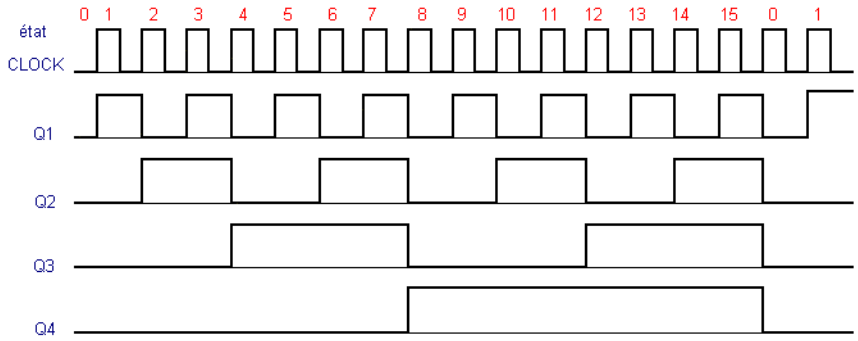
La solution pour pouvoir s'affranchir de ces états perturbateurs est de rendre le système **totalement synchrone**. Pour cela, il est indispensable que toutes les bascules reçoivent le **même signal d'horloge**.

La synthèse d'un compteur synchrone consiste alors à trouver la commande nécessaire des entrées des bascules (T, D ou JK) pour obtenir la séquence déterminée sur les sorties.



Compteur synchrone modulo 2^n

Par exemple, on souhaite réaliser un compteur modulo 16, tel que décrit par le chronogramme suivant :



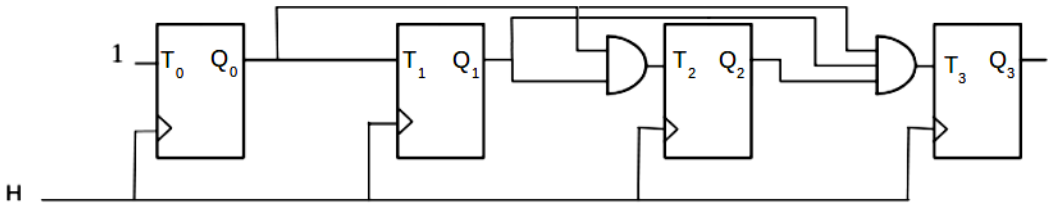
On peut remarquer que le bit de poids faible change à tous les coups d'horloge et qu'un bit quelconque change lorsque tous les bits de poids plus faible sont égaux à 1.

La réalisation à partir de bascules T ou JK est alors simple. Sachant que sur une bascule T (ou JK), il y a une inversion de la sortie pour $T = 1$ ($JK = 11$), on peut en déduire les entrées de chacune des bascules.

$$T_0 = 1 \qquad T_1 = Q_0 \qquad T_2 = Q_0 \cdot Q_1$$

$$T_n = Q_0 \cdot Q_1 \cdot \ldots \cdot Q_{n-1}$$

On obtient alors la structure suivante :

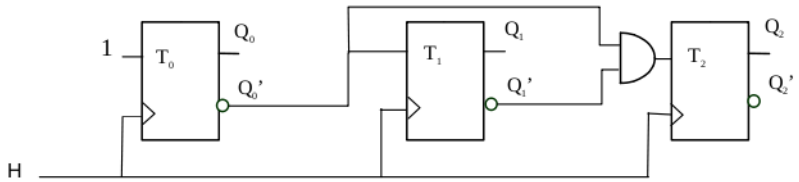


Le raisonnement fait précédemment avec des bascules T (ou JK) peut être mené à l'identique avec des bascules D.

De même, il est possible de synthétiser des compteurs ayant des **modulo différents d'une puissance de deux** (par exemple les compteurs décimaux ou modulo 10). Ce sont les équations des entrées des bascules qui vont changer.

4.1.3 Décompteurs synchrones

On peut montrer qu'en utilisant les sorties complémentées des bascules, il est possible d'obtenir un fonctionnement en décompteur. La structure d'un **décompteur modulo 8** est alors la suivante :



4.1.4 Diviseurs de fréquence

Les compteurs sont aussi utilisés pour **diviser la fréquence** d'une horloge. Cela permet de **ralentir la cadence** d'un système pour l'adapter à une fréquence voulue.

La sortie p d'un compteur à pour période :

$$T_p = 2^p \cdot T_{CLK}$$

où T_{CLK} est la période de l'horloge principale du système.
Ainsi, dans le domaine fréquentiel on obtient :

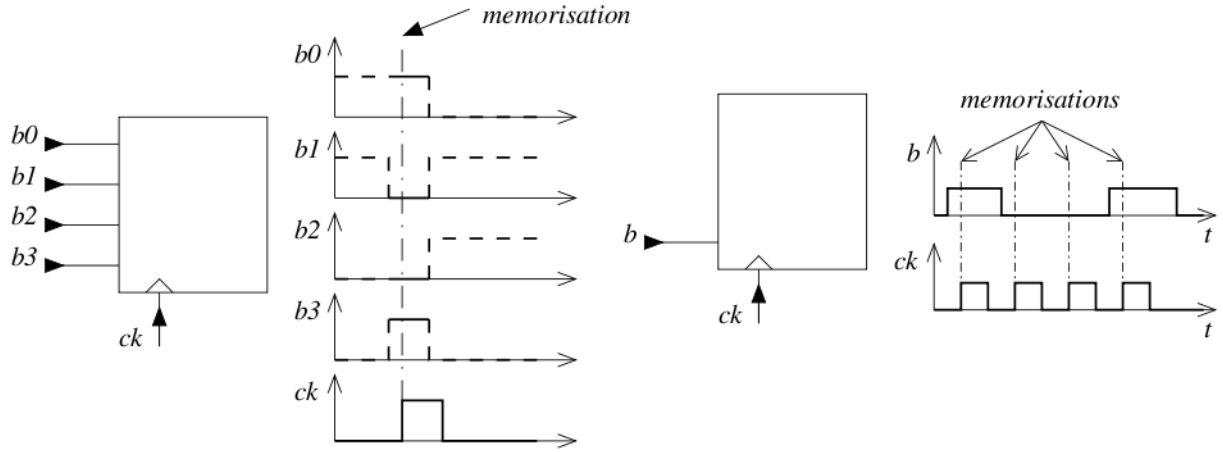
$$F_p = \frac{F_{CLK}}{2^p}$$

4.2. Registres

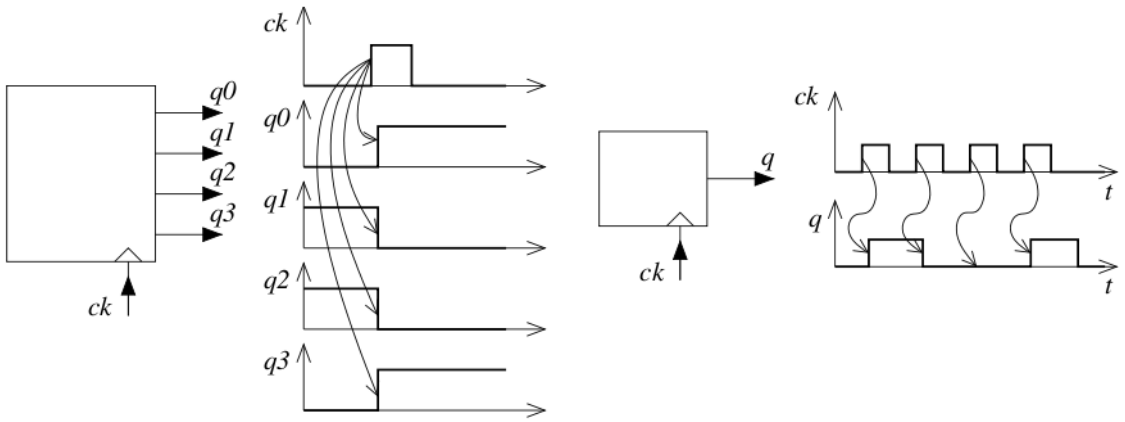
Un registre est une **association de** *n* **bascules** utilisées conjointement pour mémoriser les *n* bits d'un mot binaire.

Les *n* bascules ont une horloge commune. Les **bascules D** sont les plus utilisées dans les registres.

Il existe deux **modes de chargement en entrée** :
- en **parallèle** : un fil séparé est prévu pour chaque entrée D des bascules ;
- en série : un seul fil est prévu pour l'entrée D de la première bascule.



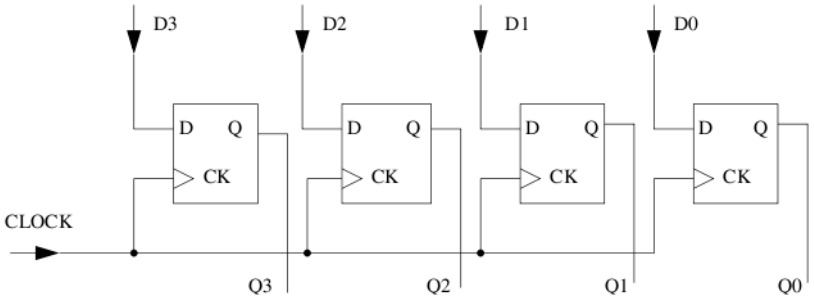
Il existe deux **modes de lecture en sortie** :
- en **parallèle** : un fil séparé est prévu pour chaque sortie Q des bascules ;
- en série : un seul fil est prévu pour la sortie Q de la dernière bascule.



Il existe alors plusieurs combinaisons possibles d'entrée et de sortie. Selon ces combinaisons, les registres portent des noms différents :
- **registre à décalage** : série / série
- conversion **série/parallèle** de données
- conversion **parallèle/série** de données
- **registre d'état** / mémorisation : parallèle/parallèle

4.2.1 Registres de mémorisation

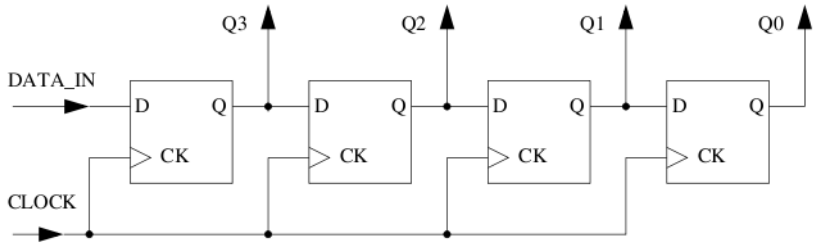
Le **registre de mémorisation** est le registre élémentaire. Il est constitué d'une juxtaposition de bascules permettant de mémoriser un mot binaire. Ce registre est également appelé registre à entrées parallèles.



4.2.2 Registres à décalage

Le registre à décalage est une association de bascules permettant de décaler un mot binaire, à droite ou à gauche.

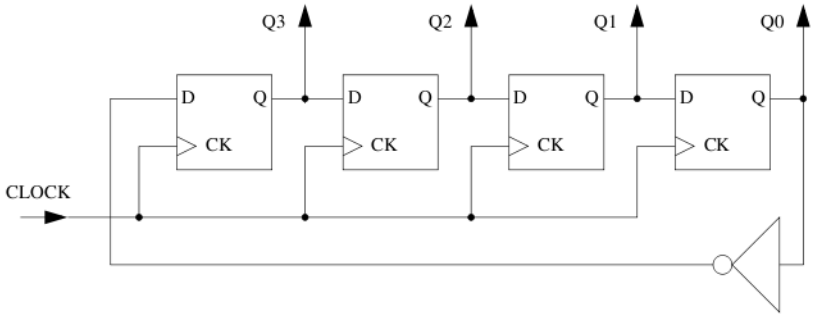
Dans tous les cas, l'information est disponible intégralement en *n* coups d'horloge après le chargement pour un mot de *n* bits.

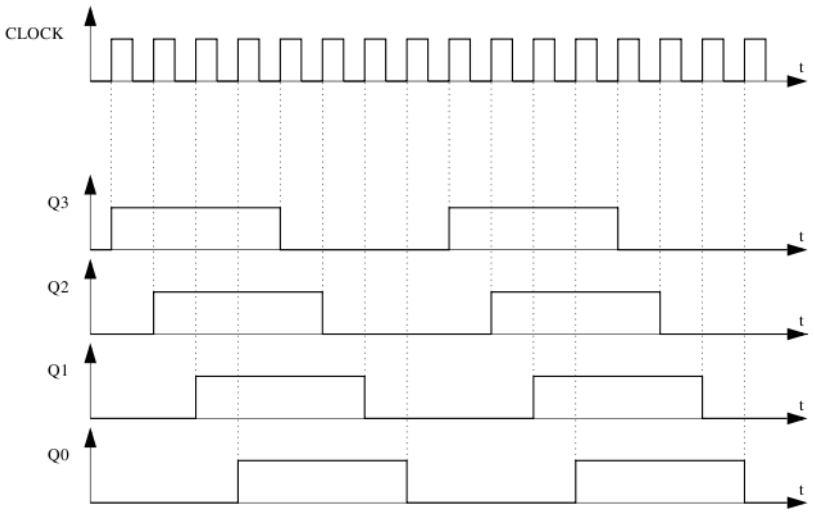


4.2.3 Compteur Johnson

C'est un registre à décalage dont la dernière sortie est rebouclée sur l'entrée via un inverseur.

Une séquence principale est initialisée dans le dispositif et tourne en permanence dans cette structure.




CLOCK
t
Q3
t
Q2
t
Q1
t
Q0
t

5. Machine à états

Les machines à état permettent de décrire des systèmes séquentiels dont l'**évolution** est plus **complexe** que les compteurs ou les registres.

Il est remarquable de constater que le concept relatif aux automates (au sens machines à état) se retrouvent désormais dans des applications diverses :

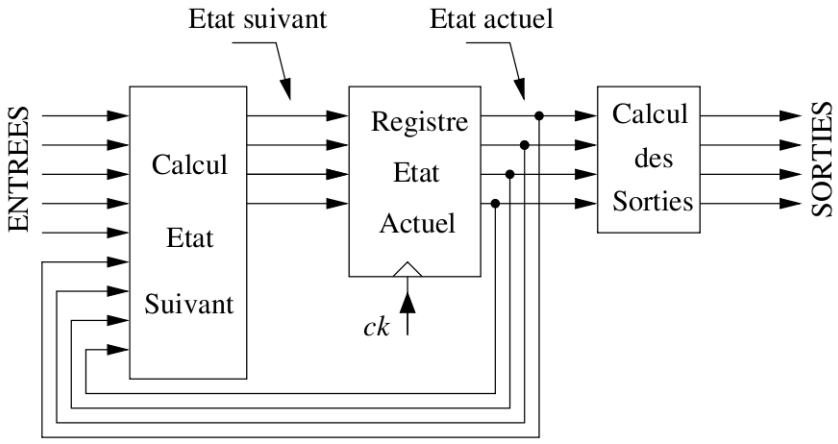
- circuits numériques ;
- automatismes industriels ;
- processeurs ou microcontroleurs ;
- prgrammes informatiques.

5.1. Modèles de machine à états

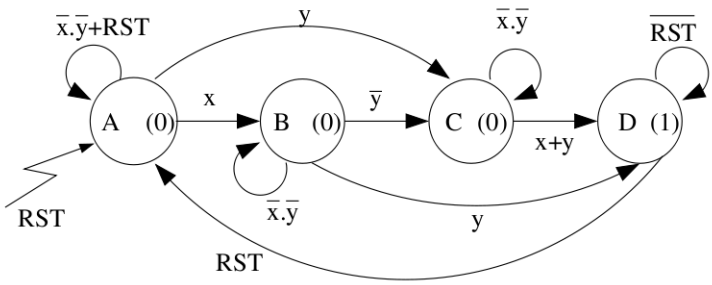
Pour représenter ces automates, qu'ils soient matériel ou logiciel, il existe deux architectures différentes : la machine de **MOORE** (synchrone) et la machine de **MEALY** (asynchrone).

5.1.1 Modèle de Moore - synchrone

Dans une **machine de Moore**, la sortie ne dépend que de l'état de la machine. Les sorties sont alors synchrones avec les transitions d'état et les fronts d'horloge.



Voici un exemple de diagramme d'états que l'on pourrait associé à une machine de Moore :

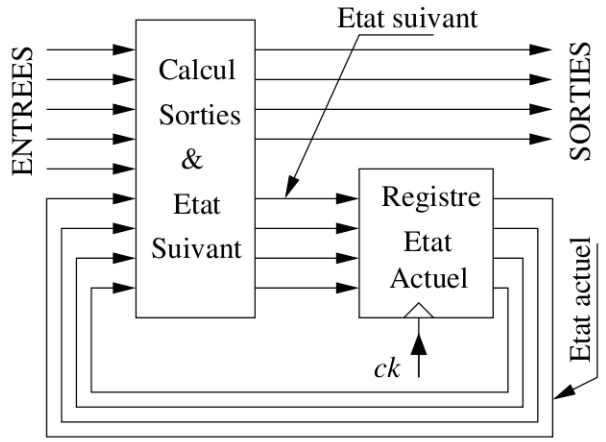


Dans une telle machine, les sorties étant fonction exclusivement de l'état du système, leurs valeurs sont indiquées dans les cercles.

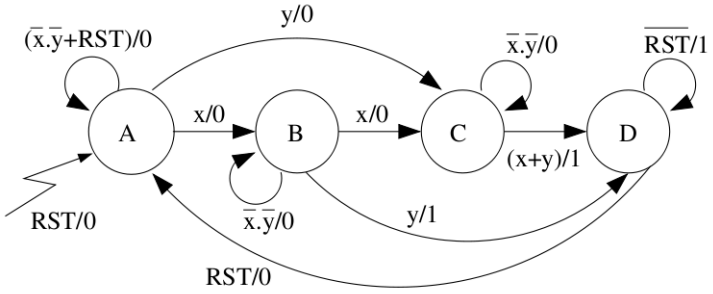
5.1.2 Modèle de Mealy - asynchrone

Dans une **machine de Mealy**, la sortie est calculée en fonction de l'état présent et de la valeur présente des entrées : les sorties peuvent alors changer immédiatement après un changement des entrées, indépendamment de l'horloge.

Ces systèmes sont alors totalement **asynchrones**. Ils sont plus rapides que les machines de Moore, mais beaucoup plus instables et difficiles à concevoir.



Voici un exemple de diagramme d'états que l'on pourrait associé à une machine de Mealy :



5.2. Conception et synthèse de machines à états

Afin d'illustrer la synthèse d'une machine à états, nous prendrons comme exemple un **détecteur de séquence**. Un tel système est très souvent utilisé autour de nous : digicode, détecteur d'erreurs CRC...

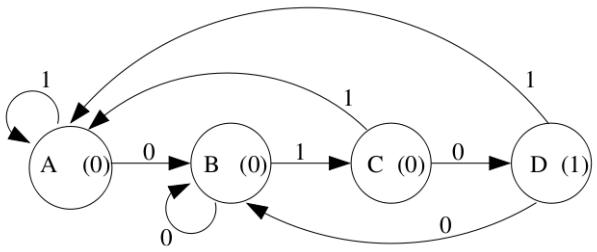
Le système à réaliser a une entrée E et une sortie S. E reçoit des bits en série, cadencés par une horloge.

Chaque fois que la séquence *"010"* se présente en entrée, la sortie S devra passer à '1' dès le dernier bit détecté, puis retourner à '0' au bit suivant, quel qu'il soit.

5.2.1 Diagramme d'états

La première étape lors de la conception d'une machine à états est la compréhension du cahier des charges, qui passe souvent par la représentation sous forme d'un **diagramme d'états**.

Dans le cas précédent, l'étude du cahier des charges amène à réaliser une machine à *4 états distincts*. On peut associer le fonctionnement décrit précédemment au diagramme d'états suivant :



Il est ensuite nécessaire d'utiliser un codage particulier pour les états, qui seront stockés dans un registre d'état.

4 états nécessitent ici **2 bits d'état** (donc 2 bascules).

On pourra utiliser un codage simple des états, par exemple le code de Gray : $A = 00$, $B = 01$, $C = 11$ et $D = 10$.

5.2.2 Synthèse structurelle

Pour pouvoir plus facilement trouver la structure complète du système, c'est à dire la structure de la fonction combinatoire de calcul de l'état suivant ainsi que celle du calcul des sorties, il est préférable de passer par la **table des transitions**.

Actuel	Suivant 0	Suivant 1	Sortie S
A	B	A	0
B	B	C	0
C	D	A	0
D	B	A	1

Actuel	Suivant 0	Suivant 1	Sortie S
00	01	00	0
01	01	11	0
11	10	00	0
10	01	00	1

Dans le cas de l'utilisation de bascules D, les valeurs des entrées D des bascules sont directement donnés par les codes de l'état suivant.

On en déduit alors (après synthèse d'un système combinatoire et simplification) les expressions de D_1 et de D_2 en fonction de Q_1 , Q_2 et E ainsi que l'expression de S.

$$D_1 = E \cdot Q1 \cdot Q2 + E \cdot Q1 \cdot Q2$$

$$D_2 = E \cdot Q1 + E \cdot Q2 + Q1 \cdot Q2$$

$$S = Q1 \cdot Q2$$

5.2.3 Synthèse comportementale (VHDL)

L'autre possibilité, à partir du diagramme d'états, est de décrire le **comportement du système** à l'aide d'un **langage de description de haut niveau**, tel que le VHDL.

La traduction du diagramme d'états précédent est donnée par la suite.

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity detect_seq is
  port
  (
    E, CLK: in STD_LOGIC;
    s:     out STD_LOGIC
  );
end detect_seq;

architecture mach_etat of detect_seq is
signal ETAT: STD_LOGIC_VECTOR(1 downto 0);

mach: process (CLK)
begin
  if (CLK'event and CLK='1') then
    case ETAT is
    when "00" =>
      if E='0' then ETAT <= "01";
      else ETAT <= "00";
      end if;
    when "01" =>
      if E='0' then ETAT <= "01";
      else ETAT <= "11";
      end if;
    when "10" =>
      if E='0' then ETAT <= "01";
      else ETAT <= "00";
      end if;
    when others =>
      if E='0' then ETAT <= "10";
      else ETAT <= "00";
      end if;
    end case;
  end if;
end process mach;

S <= '1' when ETAT = "10" else '0';

end mach_etat;
```

Notes

Notes (suite)

Notes (suite)