CIRCUITS LOGIQUES
GEL-10279

Notes de Cours

par
Dominic Grenier, Ph.D.

revue et corrigée par
Denis Laurendeau

graphisme
Gaston Guay

UNIVERSITÉ LAVAL

CIRCUITS LOGIQUES

TABLE DES MATIERES

CHAP.3 SYNTHESE DE CIRCUITS COMBINATOIRES

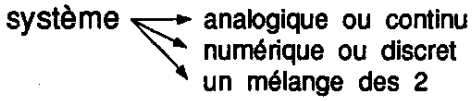
CHAP. 4 ELEMENTS SYNCHRONES

CHAPITRE 1
INTRODUCTION

1.1 - Définition

L'électronique se subdivise en deux grandes familles distinctes qui sont l'électronique analogique (linéaire ou non) et l'électronique numérique. Les circuits logiques font partie de cette dernière famille.

Un système numérique est un système dynamique qui évolue d'une façon discrète dans le temps et où les amplitudes des signaux ne peuvent prendre qu'un nombre fini de valeurs par opposition au système analogique. Un système peut être défini mathématiquement comme un opérateur unique qui transforme des conditions d'entrée en des sorties spécifiques.



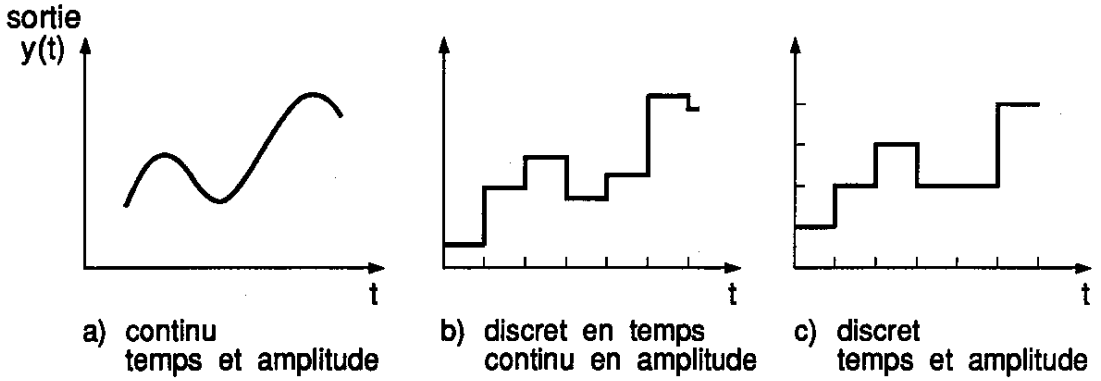
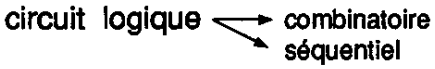


Figure 1.1 - Évolution des signaux dans un système.

Les systèmes numériques sont formés d'éléments plus simples qu'on nomme les circuits logiques. Ces derniers se partagent essentiellement en deux groupes ou classes : Les circuits combinatoires et les circuits séquentiels.

Les circuits combinatoires sont sans mémoire, leurs sorties ne dépendent donc que des entrées au moment de la mesure. Par contre, les sorties des circuits séquentiels sont fonction non seulement des entrées mais, en plus, des anciens états des circuits, soit de l'histoire passée des variables.



1.2 - La technologie

Les circuits logiques sont formés à partir d'éléments électroniques simples, reliés entre eux dans le but de réaliser une fonction et intégré (IC). Évidemment, la théorie entourant les circuits logiques électroniques s'applique à des systèmes pneumatiques et fluidiques.

Or, depuis plusieurs années, la technologie des circuits intégrés s'est constamment développée de sorte que l'intégration, la rapidité d'exécution, les puissances dissipées et le coût ont de beaucoup diminué. Cette technologie nouvelle a influencé les critères de design tels que, par ordre de priorité :

* le fonctionnement spécifié
* la fiabilité
* la facilité de fabrication et d'entretien
* le coût minimum

Parlant de coût, on remarque que le coût des circuits intégrés est devenu une fraction faible du coût total d'un système logique. Par contre, on vérifie par expérience que le coût d'un système logique demeure proportionnel au nombres de boîtiers impliqués dans le design. La recherche d'un circuit à coût minimum en boîtiers devient donc un critère d'une certaine importance au point de vue pédagogique même si la règle a toutefois des lacunes (certaines conditions supplémentaires devraient y être rajoutées pour raffermir la validité de la règle du coût minimum). Malgré tout, cette règle semble très appropriée notamment pour la logique câblée, incluant les circuits à très large niveau d'intégration (VLSI). En conséquence, il est souvent avantageux de remplacer de nombreux boitiers à faible niveau d'intégration par quelques uns du type moyen ou large niveau.

1.3 - Niveau d'intégration

Le niveau d'intégration s'interprête de plusieurs façons. Évalué selon le nombre de transistors constituant la pastille de silicium, le niveau d'intégration illustre la complexité de la fonction réalisée par le circuit intégré. Dans le jargon de l'électronique, on parle de quatre familles de niveau d'intégration qu'on appelle souvent par abréviation.

SSI : (Small Scale Intégration)
éléments simples réalisant une opération logique
de base moins d'une cinquantaine de transistors

MSI : (Medium Scale Intégration)
éléments moyennement complexes tels les bascules,
les régistres, les compteurs, les multiplexeurs, les décodeurs,
etc... moins de 300 à 500 transistors

LSI : (Large Scale Intégration)
éléments de grande complexité allant des mémoires vives jusqu'aux microprocesseurs simples et ce, en passant par les ROM's, PLA's de base, etc... plus de 500 transistors mais moins de 10000

VLSI : (Very Large Scale Intégration)
éléments les plus complexes tels les microprocesseurs de haute gamme ou les PLA's complexes. (500,000 transistors dans le micropresseur du HP9000)
(plus de 1 million de transistors dans le Megabit RAM dynamique M5M41000S de Mitsubishi)

1.4 - Système de nombre

1.4.1 - Numérotation de position

Un système de nombre consiste en un assemblage mathématique de plusieurs chiffres, chaque chiffre contient un symbole faisant partie d'un ensemble et représente un nombre. Le nombre de symboles dans l'ensemble représente la base du système.

La construction du nombre à partir des symboles suit quelques règles qui favorisent l'implantation d'opérations mathématiques. On a donc :

$$(N)_r = \left[(\text{partie entière}) \bullet (\text{partie fractionnelle})\right]_r$$

point radical

La construction la plus usuelle est basée sur le poids du chiffre variant avec sa position dans l'expression du nombre. C'est la numérotation de position.

$$(N)_r = \underbrace{(a_{n-1}\, a_{n-2} \ldots a_1 a_0}_{\text{partie entière}} \bullet \underbrace{a_1 a_{-2} \ldots a_{-m})_r}_{\text{partie fractionnaire}}$$

point radical

avec N = nombre à représenter
r = base du système
a_i = chiffre
un des symboles de l'ensemble
n = nombre de chiffres dans la partie entière de la représentation
m = nombre de chiffres dans la partie fractionnaire de la représentation

Prenez note que r étant le nombre de symboles dans l'ensemble, alors :

$0 \leq a_i < r$

Par exemple, la base décimale (r=10) compte 10 symboles soient les dix chiffres s'échelonnant de 0 à 9. Evidemment, pour une base supérieure à 10, on doit créer de nouveaux symboles mais il est de mise de se servir des lettres alphabétiques. Quant au poids des symboles dans la présentation, il augmente en s'éloignant du point radical dans la partie entière et vice-versa dans la partie fractionnaire

$ 3675.40 indique :		3	fois	1000	dollars
	plus	6	fois	100	dollars
	plus	7	fois	10	dollars
	plus	5	fois	1	dollar
	plus	4	fois	0.1	dollar (10¢)

Le principe est identique à celui adopté par les "Nonos" pour leur système monétaire en "fous" (φ)

φ 1101.01 indique :		1	fois	8	fous
	plus	1	fois	4	fous
	plus	1	fois	1	fou
	plus	1	fois	¼	fou

dans la base binaire (r=2)

1.4.2 - Notation polynominale

La représentation d'un nombre axée sur le poids du chiffre dépendamment de sa position suggère une notation simple du nombre. Cette notation indique, de plus, le poids du chiffre et ainsi facilite la quantification du nombre exprimé dans une autre base.

$$(N)_r = \sum_{j=-m}^{n-1} a_j r^j$$

Ainsi, on remarque que la pondération du chiffre est fonction du facteur (r^j). les deux exemples suivants illustrent le principe :

exemple #1: système binaire (r=2)

$$(N)_2 = (1101.101)_2$$
$$= 1x2^3+1x2^2+1x2^0+1x2^{-1}+1x2^{-3}$$
$$= (13\ 5/8)_{10}$$

exemple #2: système hexadécimal (r=16)
pour ce faire, on crée 6 nouveaux symboles

$(A)_{16} = (10)_{10}$
$(B)_{16} = (11)_{10}$
$(C)_{16} = (12)_{10}$
$(D)_{16} = (13)_{10}$
$(E)_{16} = (14)_{10}$
$(F)_{16} = (15)_{10}$
$(N)_{16} = (4C0A.7D)_{16}$
$= 4x16^3 + 12x16^2 + 10x16^0 + 7x16^{-1} + 13x16^{-2}$
$= (19466\ 125/256)_{10}$

1.5 - Système binaire

Pour des raisons techniques essentiellement, la technologie des circuits logiques actuelle emploie des éléments binaires. C'est donc dire que la base binaire (r=2) est au coeur même de l'arithmétique des ordinateurs par exemple.

Une analyse plus approfondie révèle que le choix de cette base offre des particularités intéressantes. En outre, l'ensemble binaire, formé des éléments 0 et 1, est un corps de Galoi avec les 2 opérateurs + et · diadiques. On peut donc appliquer la théorie des corps à l'ensemble binaire. De plus, on démontre que la logique des décisions présente deux états seulement (le vrai et le faux) qui peuvent être codés par 1 et 0 respectivement.

On doit donc connaître la représentation des nombres dans cette base lorsqu'on tient compte du signe. Pour ce faire, il existe 3 méthodes :

représentation des nombres avec signe →
- valeur absolue et bit de signe
- complément restreint
- complément vrai

Il est à noter qu'un bit représente un chiffre dans la base binaire ; qu'un octet contient 8 bits ; que dans les 3 méthodes de représentation des nombres avec signe, il faut un bit de plus que la représentation de la valeur absolue du nombre maximal qu'on veut représenter dans la base binaire.

1.5.1 - Valeur absolue et bit de signe

Cette méthode de représentation d'un nombre avec signe est sans doute la plus simple. La valeur absolue du nombre occupe les positions identiques mais le bit de poids fort (dans le jargon, le MSB pour "most significant bit") indique le signe du nombre selon la convention :

0 → nombre positif
1 → nombre négatif

Cette méthode permet de représenter les nombres s'étendant sur la plage

$-(2^{n-1}-1)$ à $(2^{n-1}-1)$

ou

n = nombre de bits de la représentation

exemples: les nombres décimaux entre −7 et 7 peuvent se représenter sur 4 bits
3 bits pour la valeur absolue
1 bit de signe

$(5)_{10} = (0101)_{2,\,va+s}$

$(-5)_{10} = (1101)_{2,\,va+s}$

NSB — base 2 avec valeur absolue et signe

1.5.2 - Complément restreint (appelé aussi "complément 1")

Aussi connue sous le nom de "(r−1)'s complément", cette méthode s'applique sur l'ensemble des bases. (r indiquant la base utilisée). Encore ici, le MSB indique le signe du nombre représenté.

MSB
0 → nombre positif
1 → nombre négatif

Un nombre positif se représente, en complément restreint, de la même façon qu'en valeur absolue avec bit de signe. La différence apparait avec les nombres négatifs. Ici, il faut effectuer une petite transformation puis représenter le résultat de la transformation comme s'il n'y avait pas de signe.

Soit M la valeur absolue du nombre négatif N représentée sur n bits. Alors

$(N)_{2,cr} = (2^n-1)-M$

est la transformation à faire sur la base binaire (plus généralement $(r^n-1)-M$).

Une astuce simple permet de faire la conversion directement au nombre absolu de n bits vers la représentation du nombre négatif en complément restreint : prendre chaque bit tour à tour et le complémenter (1 → devient 0 et 0 → devient 1). Cette astuce n'est cependant valide que pour la base binaire. Cette méthode permet

de représenter les nombres s'étendant sur la plage identique à celle de la valeur absolue avec bit de signe et pour ces 2 méthodes, vous pouvez vérifier que le nombre (0) se code de 2 façons.

exemples: les nombres décimaux de −7 et 7

$(0)_{10} = (0000)_{2,cr}$ ou $(1111)_{2,cr}$
$(6)_{10} = (0110)_{2,cr}$
$(-6)_{10} = (1001)_{2,cr} \leftarrow (2^4-1)-6 = 9 = 1001$
$(7)_{10} = (0111)_{2,cr}$
$(-7)_{10} = (1000)_{2,cr}$

1.5.3 - Complément vrai (appelé aussi "Complément 2")

Aussi connue sous le nom de "(r)'s complément" ou complément 2 dans la base binaire, cette méthode est de loin la plus utilisée chez les ordinateurs. Pour une troisième fois, la MSB indique le signe du nombre représenté.

0 → nombre positif
1 → nombre négatif

Cette méthode s'apparente facilement avec celle du complément restreint à la différence près de la transformation qui devient :

$$\underbrace{(N)_{2,cv}}_{\text{base 2, complément vrai}} = (2^n) - M$$

avec

M = valeur absolue du nombre négatif N
n = nombre de bits de la représentation

Ainsi, pour trouver le complément vrai, il s'agit de rechercher le complément restreint selon l'astuce proposée précédemment et d'additionner 1. Avec cette méthode, un seul zéro et la plage des nombres possiblement représentés sur n bits s'étend de

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

exemples: les nombres décimaux de −8 à 7 peuvent être représentés sur 4 bits

$(0)_{10} = (0000)_{2,cv}$
$(7)_{10} = (0111)_{2,cv}$
$(-7)_{10} = (1001)_{2,cv}$
$(-8)_{10} = (1000)_{2,cv}$
$(4)_{10} = (0100)_{2,cv}$
$(-4)_{10} = (1100)_{2,cv}$

1.6 - Système hexadécimal

Il s'agit ici d'un système très répandu dans le domaine de l'informatique et pour cause, car un digit peut représenter 4 bits ($2^4 = 16$) soit la moitié des bits d'un octet. C'est donc dire que 2 codes hexadécimaux suffisent pour donner la valeur d'un octet.

$(0)_{16} = (0000)_2$
$(1)_{16} = (0001)_2$
$\vdots$
$(9)_{16} = (1001)_2$

$(A)_{16} = (1010)_2 = (10)_{10}$
$(B)_{16} = (1011)_2 = (11)_{10}$
$(C)_{16} = (1100)_2 = (12)_{10}$
$(D)_{16} = (1101)_2 = (13)_{10}$
$(E)_{16} = (1110)_2 = (14)_{10}$
$(F)_{16} = (1111)_2 = (15)_{10}$

$(AB)_{16} = (1010\ 1011)_2$

CHAPITRE 2

NOTIONS ÉLÉMENTAIRES DE LOGIQUE BINAIRE

2.1 - Notion de corps

L'ensemble binaire constitué des éléments {0,1} est, rappelons-le, le plus petit corps de Galoi. Il répond donc aux exigences suivantes :

F = {0,1} et les deux opérateurs + et · tels que $\forall a,b,c \in F$

F, +, · est un anneau commutatif d'où

a+b = b+a, ab = ba (commutativité)
(a+b)+c = a+(b+c), (ab)c = a(bc) (associativité)
a(b+c) = ab+ac (distributivité)

F*, +, · est un groupe d'où

F* = F − {0}

existence d'une identité (I) tel que aI= Ia=a
existence d'un inverse (-a) tel que a(-a)= I

En appliquant ces définitions, on remarque que les corps les plus usuels sont les réels ou les complexes. Mais ce sont des corps d'ordre infini. En prenant des opérations "modulo" on parvient à construire des corps d'ordre fini que sont les corps de Galoi dont le corps binaire. On définit alors les opérations + et ·

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

2.2 - Algèbre de Boole

Découlant de la théorie des corps, l'algèbre développée par George Boole dans son travail "An investigation of the laws of thought on which are founded the mathematical theories of logic and probability" (1854) fait le pont entre la logique des décisions (ou du langage) et celles des circuits logiques. L'algèbre de Boole devient donc l'outil mathématique servant à l'analyse des décisions à partir des propositions. Les circuits logiques fournissent les outils physiques pour matérialiser les opérateurs.

ALgèbre de Boole → outils mathématiques
Circuits logiques → outils physiques

Les deux opérateurs spécifiés par Boole diffèrent de ceux découlant de la théorie des corps pour se rapprocher davantage des opérateurs logiques "ET" et "OU". En fait, seule l'opération + est modifiée légèrement.

+	0	1
0	0	1
1	1	1

Cependant, la modification de l'opérateur + oblige, pour garder la théorie des corps valide, l'ajout d'un opérateur monadique appelé "inverse" qui réalise la négation de la proposition.

opérateurs logiques → ET (·)
OU (+)
négation ($^-$)

Attention : Pas de soustraction et de division!

$a+b = a+c \not\Rightarrow b = c$
$ab = ac \not\Rightarrow b = c$ *($\not\Rightarrow$: n'implique pas)*

On peut associer l'algèbre de Boole à la théorie des ensembles décrite par les diagrammes de Venn. Mais ces derniers deviennent vite encombrants d'où l'importance de l'utilisation des théorèmes algébriques de Boole :

diagramme de Venn → union = OU (+)
intersection = ET (·)
complément = inverse ($^-$)

théorie des ensembles	algèbre de Boole	logiques des décisions
U (ensemble universel)	1	vrai
ϕ (ensemble vide)	0	faux
V	+	ou
$\wedge$	·	et
$\bar{A}$	$\bar{A}$	négation

2.2.1 - Fonction logique et table de vérité

Avant de dresser la liste des théorèmes de Boole, il vaut mieux définir ce qu'est une fonction logique, car le rôle de ces théorèmes est de permettre une simplification de l'expression de la fonction logique.

Supposons plusieurs propositions qui sont indépendantes les unes des autres. Une décision prise dépendant d'un arrangement de ces propositions fait que la décision est fonction logique des propositions. Plus simplement, on dit que "décision" est une fonction logique et que les propositions sont les variables indépendantes du problème (notées ultérieurement V.I.).

Dans l'expression orale de la fonction logique, on se sert communément des opérateurs que sont les locutions ni (ou non), et, ou sans trop de préséance entre eux. Ceci nous conduit parfois vers des ambiguïtés. La mathématique, elle, exige plus de rigueur. On obtient donc la préséance suivante :

préséance
forte → faible
$(^{-})$ → $(\cdot)$ → $(+)$

Voici un exemple d'une fonction logique. Un bon étudiant s'interroge pour savoir s'il est sage de sortir ce soir. Il doit prendre une décision en fonction des quatre propositions indépendantes suivantes :

A = assez d'argent
B = devoirs pas complétés
C = auto à papa disponible
D = transport en commun en grève

D'où on déduit que l'étudiant sortira si A est vraie, B est fausse et que C *est vraie ou D est fausse.*

$$\text{sortir} = A\bar{B}(C+\bar{D})$$

Il est moins évident de rechercher la condition à respecter pour que l'étudiant ne sorte pas. Mais, l'étude des théorèmes Booléens fournit la réponse.

$$\overline{\text{sortir}} = \bar{A}+B+\bar{C}D$$

La construction de l'expression logique dit que (sortir) est une fonction logique des variables indépendantes A, B, C et D.

Dans la plupart des cas, le problème se complique de sorte qu'il n'est pas facile de sortir une expression simplifiée de la fonction logique. C'est là qu'intervient l'algèbre de Boole. Pour démarrer le problème, on dresse une table assignant la valeur de la fonction pour chacune des combinaisons des variables indépendantes. Cette table s'appelle la table de vérité et elle est unique pour la fonction donnée. Ainsi, la table de vérité définit complètement la fonction logique.

Exemple: table de vérité
3 variables indépendantes A, B, C

F(A,B,C)

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

vérifiez que l'équation
$F(A,B,C) = A \cdot B + \bar{A}C + AC$
répond à cette table de vérité

avec les simplifications possibles, on obtient
$F(A,B,C) = A \cdot B + C$

2.2.2 - Théorèmes de Boole

L'algèbre de Boole se greffe autour d'une liste de 22 théorèmes fondamentaux. Il est intéressant de noter que la moitié (ou presque) d'entre eux sont le dual de l'autre : en remplaçant l'opérateur "et" par l'opérateur "ou" vice-versa, et en changeant les "0" par les "1" et vice-versa : on retrouve l'autre portion des théorèmes du tableau.

Ceci implique la notion de principe de dualité $\left\{ \begin{matrix} + \longrightarrow \cdot \\ \cdot \longrightarrow + \end{matrix} \right\}$ et $\left\{ \begin{matrix} 0 \longrightarrow 1 \\ 1 \longrightarrow 0 \end{matrix} \right\}$

Identité			
1.	$0 \cdot A = 0$	2.	$1 + A = 1$
3.	$1 \cdot A = A$	4.	$0 + A = A$
5.	$A \cdot A = A$	6.	$A + A = A$
7.	$A \cdot \bar{A} = 0$	8.	$A + \bar{A} = 1$
commutativité, associativité, distributivité			
9.	$AB = BA$	10.	$A + B = B + A$
11.	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	12.	$A + (B + C) = (A + B) + C$
13.	$A \cdot (B + C) = AB + AC$	14.	$A + BC = (A + B) \cdot (A + C)$
absorption, adjacence logique			
15.	$A + AB = A$	16.	$A(A + B) = A$
17.	$AB + A\bar{B} = A$	18.	$(A + B)(A + \bar{B}) = A$
lois de DeMorgan			
19.	$\overline{AB} = \bar{A} + \bar{B}$ $\overline{AB...M} = \bar{A} + \bar{B} + ... \bar{M}$	20.	$\overline{A + B} = \bar{A}\bar{B}$ $\overline{A + B + ... M} = \bar{A}\bar{B}...\bar{M}$
autres théorèmes			
21.	$A + \bar{A}B = A + B$	22.	$A(\bar{A} + B) = AB$

Tableau 2.1 - Théorèmes booléens

Plusieurs théorèmes sont évidents ; d'autres se démontrent à partir de ceux de base tel le théorème 15.

$A + AB = A \cdot 1 + A \cdot B$ (théorème 3)
$= A (1 + B)$ (théorème 13)
$= A$ (théorème 2)

Exemple: simplifier l'expression
$F = (A+B+C)(\bar{A}+B+C) + AB + AC$

faire la distribution des parenthèses
$F = (A\bar{A}+AB+AC+\bar{A}B+BB+BC+\bar{A}C+BC+CC) + AB + AC$

0 th.7 B th.5 C th.5

$F = AB+AC+\bar{A}B+B+BC+\bar{A}C+BC+C + AB + AC$

AB th.6 AC th.6 BC th.6

$F = AB+AC+\bar{A}B+B+BC+\bar{A}C+C$

B th.17 C th.17 B th.15

$F = B+C+B+C$

B th.6 C

$F = B+C$

Note : AB, AC et BC sont considérés comme des variables simples pour appliquer le théorème 6. Ces groupements de variables représentent la variable A du théorème.

On peut toujours passer par d'autres simplifications, mais on arrive pratiquement toujours au même résultat.

2.3 - Les familles logiques

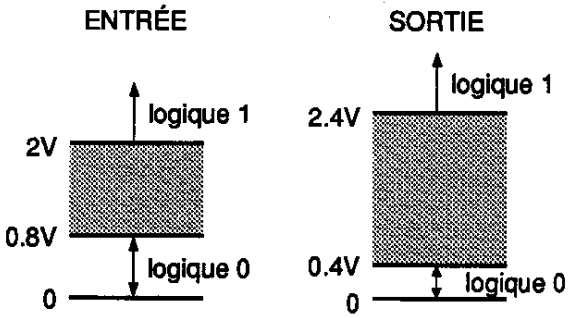
Pour réaliser une fonction logique matérielle (et non seulement sur le papier), on a créé des éléments qui réalisent les fonctions logiques les plus courantes. Les éléments sont fabriqués selon diverses technologies. On a cherché à standardiser les technologies, méthodes de conception et de fabrication des circuits intégrés selon :

- la tension d'alimentation
- les niveaux de fonctionnement
- la vitesse
- la puissance dissipée
- l'immunité aux bruits
- le fan-out (sortance)
- etc...

Lorsque plusieurs éléments, les circuits intégrés logiques de complexités diverses sont fabriqués avec la même technologie et qu'ils respectent les standards, ils constituent une famille.

exemple : famille TTL (transistor-transistor logic)

* alimentation 0-5V ± 5%
* niveaux de fonctionnement



* immunité aux bruits de 0.4V
* 5 sous-familles

(00	S00	LS00	AS00	ALS00)	
10	3	9.5	1-4	3-10	délai de propagation (ns)
10	20	2		< 3 pour f< 10 Mhz	puissance dissipée par porte (mW)

* fan-out inférieur à 10

Les autres familles logiques très répandues sont le CMOS (Complementary Metal-Oxyde Semiconductor), le ECL (Emitter Coupled Logic) dont les caractéristiques sont transposées dans l'appendice A. En général, on admet que 2 technologies principales existent utilisant les transistors bipolaires dans une et les transitors à effet de champ dans l'autre. Les familles bipolaires (TTL, ECL, I^2L...) sont plus rapides et dissipent plus de puissance. Les familles à effet de champ (CMOS, NMOS, HMOS...) peuvent réaliser des fonctions logiques demandant une grande intégration.

2.4 - Opérateurs logiques de base

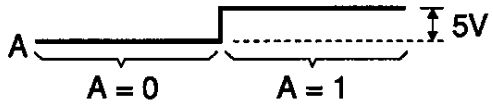
*L'algèbre de Boole implique 2 opérateurs diadiques de base (+ et ·) accompagnés d'un opérateur monadique, la négation ($\bar{}$). Ainsi, quelle que soit la famille, il est nécessaire de créer des circuits réalisant les opérateurs fondamentaux *(les portes logiques) dans chacune des familles. Les portes logiques sont donc des opérateurs booléens matériels qui permettent la transposition d'une équation logique sur papier à sa réalisation électronique. Les deux états logiques (1 et 0) sont traduits sous forme de niveaux de tension (voltage).*

exemple : TTL

1 = 5V ou V_{cc} → tension d'alimentation

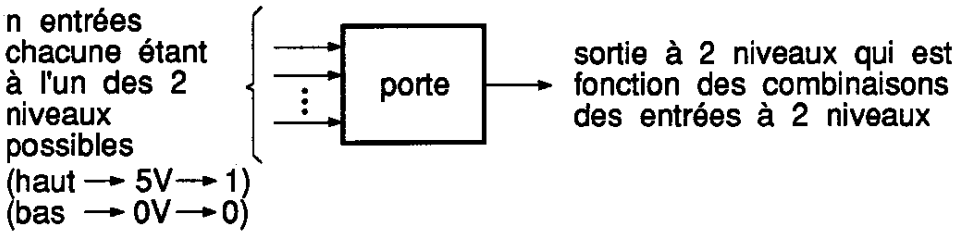
0 = 0V ou GND → référence

logique	booléen	TTL
vrai	1	5V
faux	0	0V



Définition de porte logique :	*Une porte logique est une composante matérielle à plusieurs entrées et une sortie. Le niveau de la sortie (2 possibles) est fonction de la combinaison des niveaux des signaux appliqués aux entrées.*

Cette définition, nous conduit au modèle général d'une porte



2.4.1 - Symboles distinctifs

Pour une meilleure compréhension visuelle de la fonction réalisée par une porte, on est amené à définir un symbole représentant la fonction. Comme il existe deux opérateurs, il y a donc 2 symboles distincts.

Ce symbole identifie l'opérateur "ET"("AND") une porte ayant ce symbole produit un 1 à la sortie lorsqu'il y a des "1" présents simultannément sur toutes les entrées

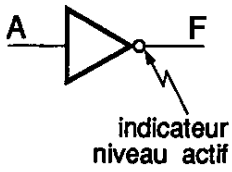
Ce symbole identifie l'opérateur "OU" ("OR") une porte ayant ce symbole produit un "0" à la sortie lorsqu'il y a des "0" présents simultannément sur toutes les entrées.

À ces deux symboles de base se greffe un indicateur qui indique le niveau d'activation. Si l'indicateur n'est pas présent à une borne, alors le signal à cette borne est actif "haut" ("High"). Dans le cas contraire, il est actif "bas" ("Low").

Cette indicateur se représente par une bulle (○).

2.4.2 - Inverseur

L'inverseur est la seule fonction logique non triviale d'une seule variable. Il inverse tout simplement le signal d'entrée.



$F = \bar{A}$

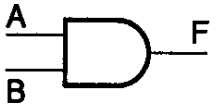
table de vérité

A	F
0	1
1	0

(cf. 7404 : 6 inverseurs par boitier)

2.4.3 - Porte "ET"

Connue sous l'appellation anglaise "AND", cette porte réalise l'opération (•).



$F = A \cdot B$

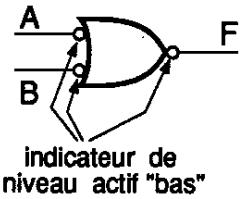
table de vérité

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Par les lois de DeMorgan, on trouve un symbole équivalent

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

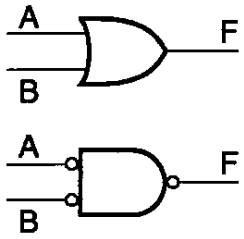
$$F = \overline{\bar{A} + \bar{B}}$$



(cf. 7408 : 4 portes "ET" à 2 variables d'entrée par boitier)

2.4.4 - Porte "OU"

Connue sous l'appellation anglaise "OR", cette porte réalise l'opération (+).



$F = A + B$

$F = A \cdot B$

table de vérité

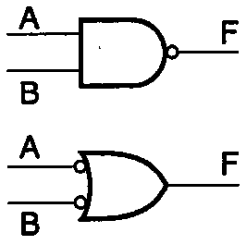
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

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

(cf. 7432 : 4 portes "OU" à 2 variables par boitier)

2.4.5 - Porte "NON ET"

Connue sous l'appellation anglaise "NAND", cette porte réalise la sortie inversée d'une porte ET. C'est la porte fondamentale de la famille TTL.



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

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

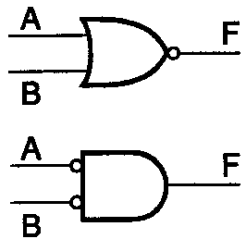
table de vérité

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

(cf. 7400 : 4 portes "NON ET" à 2 variables par boitier)

2.4.6 - Porte "NON OU"

Connue sous l'appellation anglaise "NOR", cette porte réalise la sortie inversée d'une porte "OU".



$F = \overline{A + B}$

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

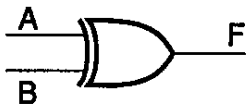
table de vérité

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

(cf. 7402 : 4 portes "NON OU" à 2 variables par boitier)

2.4.7 - Portes "EXOR" et "NEXOR"

Ces portes sont un peu spéciales : toutes les autres portes, à l'exception de celles-ci, sont disponibles avec 2, 3, 4 et parfois 8 entrées ; leur fonction dépasse les besoins de l'algèbre de Boole. La porte "EXOR" (exclusive "OR") réalise la fonction "OU exclusif" qui produit un "0" à la sortie lorsque les 2 entrées sont égales. Quant à la porte "NEXOR" (Non exclusif-OR), elle réalise la sortie inversée d'une porte "EXOR" et on l'appelle, à juste titre, la porte "identité logique". Cependant, sous forme normale, on ne la rencontre pas dans la famille TTL.



$F = A \oplus B$

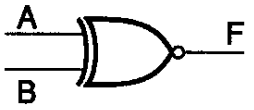
ou exclusif

table de vérité

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$F = A\bar{B} + \bar{A}B$

(cf. 7486 : 4 portes "EXOR" à 2 variables par boitier)



$F = (A \equiv B)$

identité logique

ou

$F = A \odot B$

table de vérité

A	B	F
0	0	1
1	0	0
0	1	0
1	1	1

$F = AB + \bar{A}\bar{B}$

Même si cette porte n'existe pas sous une forme normale, on la construit facilement à partir d'une porte "EXOR". D'ailleurs, on vérifie aisément que :

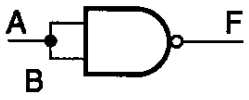
$$F = A \odot B = \overline{A \oplus B}$$
$$= \bar{A} \oplus B$$
$$= A \oplus \bar{B}$$

2.4.8 - Les portes fondamentales

Parmi toutes les portes décrites précédemment, certaines portes ou certaines combinaisons de portes semblent plus utiles car elles suffisent à synthétiser les autres. On peut tout faire :

* avec des "NAND" seulement
* avec des "NOR" seulement
* avec des "AND" et des inverseurs
* avec des "OR" et des inverseurs

exemple : réalisation d'un inverseur avec "NAND"



si A = 0 alors a = b = 0 $\quad F = \overline{a \cdot b}$, $F = 1$

si A = 1 alors a = b = 1 $\quad F = \overline{a \cdot b}$, $F = 0$

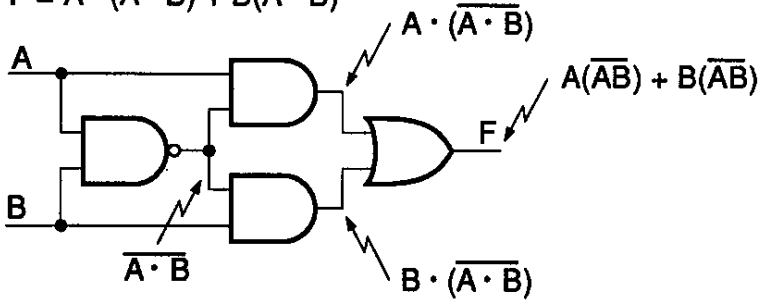
réalisation d'un "EXOR" avec "NAND"

$F = A\bar{B} + \bar{A}B$

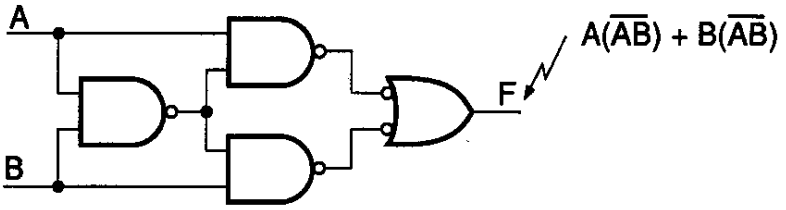
$F = A\bar{B} + \underbrace{A\bar{A}}_{0} + \bar{A}B + \underbrace{\bar{B}B}_{0}$ (théorème 7)

$F = A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$

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



Pour faire la réalisation avec des "NAND" seulement, il suffit d'ajouter la bulle d'indication du niveau d'activation à la sortie des portes "AND" (les transformant en portes "NAND") et aux entrées de la porte "OR" (la transformant en porte "NAND" aussi d'après le symbole équivalent). Les effets des bulles se cancèlent lorsque les bulles communiquent entre elles (chaque bulle inverse le niveau d'activation).



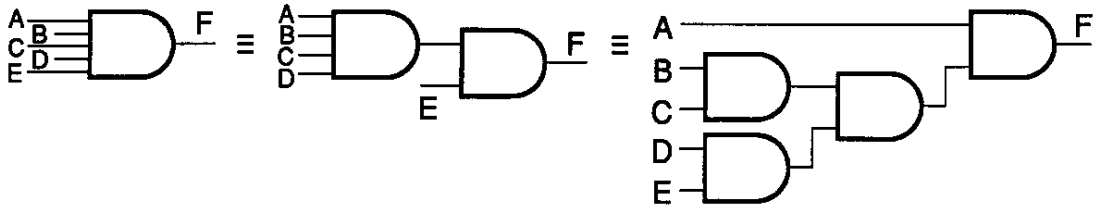
Il y a parfois avantage à utiliser le symbole équivalent d'une porte dans la représentation d'un circuit. Il allège la compréhension du circuit par inspection visuelle.

2.4.9 - Substitution de portes à plusieurs entrées

On rencontre souvent des fonctions de plus de 2 variables. Par exemple, dans l'expression $F = A \cdot B \cdot C \cdot D \cdot E$, F dépend de 5 variables. Aucune porte conventionnelle ne peut réaliser cette fonction. Seules les portes "ET" à 2,3,4 ou 8 entrées

sont disponibles. Il faut donc utiliser le théorème d'associativité pour trouver des expressions équivalentes. Par exemple,

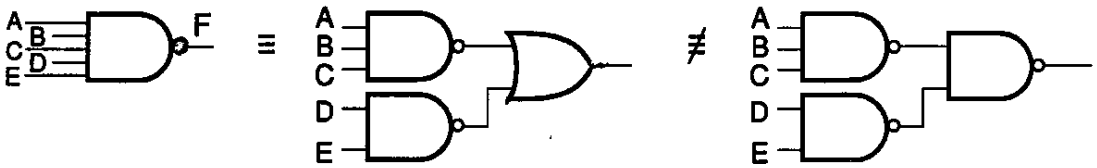
$$F = A \cdot B \cdot C \cdot D \cdot E = (A \cdot B \cdot C \cdot D) \cdot E = A \cdot [(B \cdot C) \cdot (D \cdot E)]$$



sont toutes des formes équivalentes.

Attention aux fonctions inversées ("NON-ET" , "NON-OU") ! par exemple,

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



CHAPITRE 3

SYNTHÈSE DE CIRCUITS COMBINATOIRES (i.e. sans mémoire)

La méthode classique de synthèse d'une fonction logique est d'obtenir une expression simplifiée de la fonction définie par une table de vérité. Cette expression est ensuite réalisée en utilisant des circuits intégrés simples, les portes, ou de manière moins formelle, avec des circuits plus complexes tels les multiplexeurs, décodeurs, encodeurs ou mieux les "ROM" et "PLA".

3.1 - Construction de la table de vérité

Il est possible de décrire complètement le comportement d'un circuit logique combinatoire par une expression booléenne qui indique toutes les relations entrées-sortie. Ces relations sont binaires. Les variables indépendantes d'une fonction logique sont les entrées et la variable dépendante, la sortie.

F = f(A,B,C)

A, B et C = variables indépendantes (v. ind.)
F = variable dépendante ou
valeur de la fonction " f "

Pour définir entièrement un circuit combinatoire, il faut connaître la valeur de la fonction pour toutes les combinaisons possibles des variables indépendantes. On peut obtenir ces combinaisons d'une manière systématique en construisant une table de combinaisons de la façon illustrée par la figure 3.1. On ajoute à cette table, une colonne supplémentaire indiquant la valeur de la fonction pour la combinaison des v. ind. formée par la ligne, on obtient la table de vérité.

Avec n variables indépendantes, il y a 2^n conbinaisons.

0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Figure 3.1 - Tables des combinaisons de 3 v.ind. (A, B, C). Les numéros à gauche correspondent à la valeur binaire des bits A, B et C juxtaposés.

n = 3
nombre de combinaisons = $2^n = 8$

Exemple : on veut contrôler le départ d'une course. 3 contrôleurs. Le départ a lieu si au moins 2 contrôleurs sont prêts.

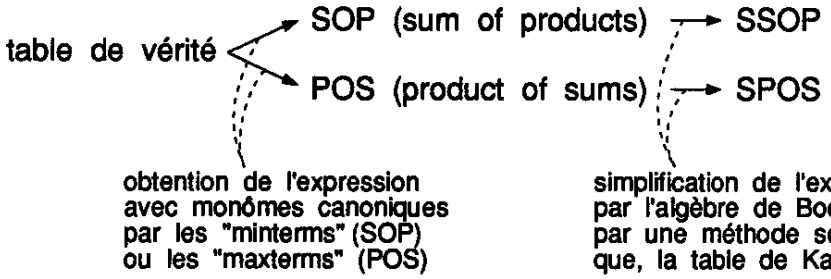
contrôleurs : Jean (J)
Suzanne (S)
Pierre (P)

l'état de préparation des contrôleurs sont les v.ind. tandis que le signal de départ est le résultat de la fonction.

J	S	P	départ
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3.2 - Passage de la table de vérité à l'expression logique d'une fonction.

Les fonctions décrites par des tables de vérité peuvent être synthétisées d'une façon systématique par leur expression booléenne de la forme d'une somme de produits (SOP) ou d'un produit de sommes (POS). Ces expressions peuvent être simplifiées pour donner les expressions SSOP ou SPOS.



Notes : *un monôme est un groupement de base dans une expression. Le monôme canonique combine l'ensemble des variables indépendantes sous forme directe ou complémentée.*

Exemple : $F = f(A, B, C, D)$

	SOP	POS
monôme :	$(\bar{A}BC)$	$(\bar{A}+\bar{B}+D)$
monôme canonique :	$(\bar{A}\bar{B}CD)$	$(A+\bar{B}+\bar{C}+\bar{D})$

Les monômes canoniques forment une base orthogonale : aucun monôme canonique ne peut être exprimé en fonction des autres. Toute fonction logique est une combinaison des monômes canoniques.

3.2.1 - Les minterms

Un minterm est un produit de toutes les variables indépendantes d'une fonction sous forme directe ou complémentée. Il est donc un monôme canonique.

Pour une fonction à n v.ind, il existe 2^n minterms indépendants soit le même nombre que de combinaisons possibles des v.ind.

Exemple : F = f (A, B, C)
n = 3

j	m_j
0	$\bar{A}\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$
2	$\bar{A}B\bar{C}$
3	$\bar{A}BC$
4	$A\bar{B}\bar{C}$
5	$A\bar{B}C$
6	$AB\bar{C}$
7	ABC

m_j = notation du minterm d'ordre j

Théorème 1a : *Un mintem est égal à 1 pour une et seulement une des combinaisons des variables indépendantes. À chaque ligne de la table de vérité correspond un seul minterm égal à 1.*

Exemple : F = f (A, B, C)
$m_4 = 1$ si seulement si A = 1
B = C = 0

Théorème 2a : *Toute fonction logique peut s'exprimer sous la forme d'une somme de minterms. On obtient alors une expression SOP.*

Exemple : F = f (A, B, C)

A	B	C	F	minterm associé
0	0	0	1	m_0
0	0	1	0	m_1
0	1	0	0	m_2
0	1	1	1	m_3
1	0	0	1	m_4
1	0	1	1	m_5
1	1	0	0	m_6
1	1	1	1	m_7

$$F = f(A, B, C) = m_0 + m_3 + m_4 + m_5 + m_7$$

Le principe est le suivant : *chaque minterm est égal à 1 pour une seule des combinaisons des v.ind. possibles. Or, une fonction F, vaut 1 pour certaines combinaisons. Il suffit donc de sommer tous les minterms égaux à 1 pour ces combinaisons.*

3.2.2 - Les maxterms

Un maxterm est une somme de toutes les variables indépendantes d'une fonction sous forme directe ou complémentée. L'expression tirée avec les maxterms est la duale de celle obtenue avec les minterms. Encore ici, il existe 2 maxterms indépendants ou n représente le nombre de v.ind.

Exemple : $F = f(A, B, C)$
$n = 3$

j	M_j
0	$A+B+C$
1	$A+B+\bar{C}$
2	$A+\bar{B}+C$
3	$A+\bar{B}+\bar{C}$
4	$\bar{A}+B+C$
5	$\bar{A}+B+\bar{C}$
6	$\bar{A}+\bar{B}+C$
7	$\bar{A}+\bar{B}+\bar{C}$

M_j = notation du maxterm d'ordre j

Théorème 1b : *Un maxterm est égal à 0 pour une et seulement une des combinaisons des variables indépendantes. À chaque ligne de la table de vérité correspond un seul maxterm égal à 0.*

Exemple : $F = f(A, B, C)$
$M_5 = 0$ si seulement si $A = C = 1$
$B = 0$

Théorème 2b : *Toute fonction logique peut s'exprimer sous la forme d'un produit de maxterms. On obtient alors une expression POS.*

Exemple : $F = f(A, B, C)$

A	B	C	F	maxterm associé
0	0	0	0	M_0
0	0	1	1	M_1
0	1	0	0	M_2
0	1	1	1	M_3
1	0	0	1	M_4
1	0	1	0	M_5
1	1	0	0	M_6
1	1	1	1	M_7

$$F = M_0 M_2 M_5 M_6$$

Le principe est de multiplier tous les maxterms associés aux combinaisons pour lesquelles la fonction, F vaut 0. Ainsi, pour toutes ces combinaisons, l'un des maxterms vaut 0, ce qui met à 0 le produit des maxterms. Il s'agit ici d'une synthèse par les 0 de la fonction.

3.2.2 - Notation et équivalence

En utilisant la technique des minterms ou des maxterms, on voit que l'on peut facilement récupérer l'information donnée dans une table de vérité pour la traduire par une équation. Cette équation est une somme de minterms (SOP) ou un produit de maxterms (POS). Pour alléger l'écriture de l'équation, on adopte la notation suivante :

$$F = \Sigma m\,(i...j)$$

ou

$$F = \Pi M\,(k...l)$$

Exemple : $F = f(A, B, C)$

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$F = m_0 + m_3 + m_5 + m_6$$

$$F = \Sigma m\,(0,3,5,6)$$

$$F = M_1 \cdot M_2 \cdot M_4 \cdot M_7$$

$$F = \Pi M\,(1,2,4,7)$$

vérification : $F = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$

$m_0 \quad m_3 \quad m_5 \quad m_6$

$\bar{F} = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}}$

par De Morgan

$\bar{F} = \overline{\bar{A}\bar{B}\bar{C}} \cdot \overline{\bar{A}BC} \cdot \overline{A\bar{B}C} \cdot \overline{AB\bar{C}}$

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

$\bar{F} = \Pi M(0,3,5,6)$

En travaillant un peu l'algèbre, on remarque qu'il existe des relations d'équivalence entre F et $\bar{F}$, minterms et maxterms.

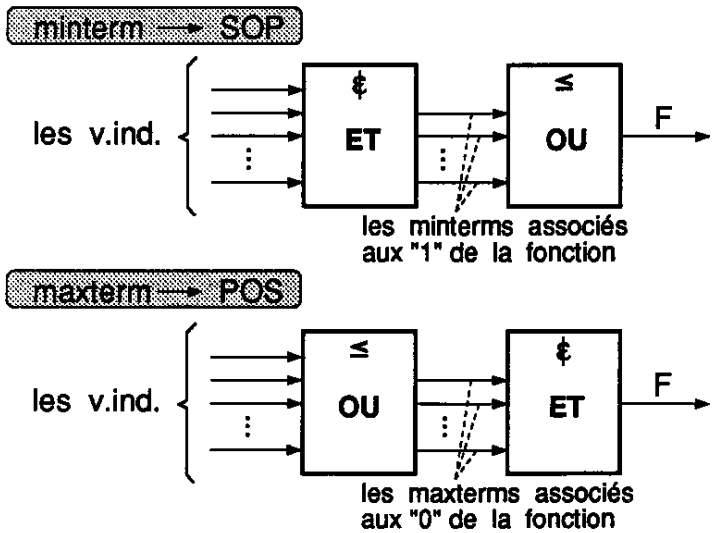
* Σ(ensemble de minterms d'ordre i) = Π(ensemble des maxterms d'ordre j, j complémentaire à i)

* $F = \Sigma$(ensemble des minterms associés aux "1" de la fonction) = Π(ensemble des maxterms associés aux "0" de la fonction)

  $\bar{F} = \Sigma$(ensemble des minterms associés aux "0" de la fonction) = Π(ensemble des maxterms associés aux "1" de la fonction)

* $\overline{m}_j = M_j$

Quelle que soit la méthode utilisée (minterm ou maxterm), l'expression obtenue ne contient que des termes canoniques. Or souvent, l'expression peut être réduite de sorte que des variables des termes canoniques sont redondantes. Il faut donc simplifier l'expression d'une fonction logique afin de réduire le nombre de variables d'un produit (d'une somme) et le nombre de termes de la somme (du produit) dans une expression du type "SOP" (POS).



Cette simplification a pour but, d'obtenir un nombre de boitiers minimum pour réaliser la ou les fonctions, procurant entre autre une surface minimale utilisée, un cablage simplifié et la possibilité d'employer des circuits intégrés plus standard (moins d'entrées possibles par porte). Évidemment, il y a toujours des compromis et, quelques fois, cette règle du nombre de boitiers minimum (aussi connue sous le nom du coût minimum) a une teneur plutôt pédagogique : celle de forcer les étudiants à la simplification et à la recherche des diverses possibilités de la synthèse.

3.5 - Simplification algébrique

Par simplification algébrique, on entend la simplification établie directement sur une expression algébrique en s'appuyant sur les quelques 22 théorèmes de l'algèbre de Boole. Seulement, l'expérience montre que 6 théorèmes suffisent, assistés, bien sûr, des identités. Ces théorèmes sont ceux d'absorption et ceux de l'adjacence logique. En vérité, 3 d'entre eux servent pour la simplification d'expressions de la forme SOP et leur dual, pour la simplification d'expressions de la forme POS.

	théorème pour **SOP**	théorème pour **POS**
I	$A + AB = A$	$A\,(A + B) = A$
II	$A + \bar{A}B = A + B$	$A\,(\bar{A} + B) = AB$
III	$AB + A\bar{B} = A$	$(A + B)\,(A + \bar{B}) = A$

Utilité des théorèmes de base

Les théorèmes I et II s'appliquent à une expression dans laquelle les termes ne sont pas canoniques ou encore à une expression qui n'est pas sous la forme d'une "somme de produit/produit de sommes". Ces théorèmes disent qu'un terme est superflu si une partie de ce terme est rencontrée dans un autre terme de l'expression; qu'une partie d'un terme est superflue si cette partie est rencontrée complémentée dans un autre terme de l'expression.

Exemple : $F = AB + BCD + \underbrace{\bar{A}CD + \bar{B}CD}_{CD(\bar{A} + \bar{B}) = CD(\overline{AB})}$

$AB + CD$ (théo. II)

Note : *AB est considéré comme une seule variable*

$F = AB + CD + BCD$

CD (théo. I)

$F = AB + CD$

Le théorème III a un intérêt particulier pour simplifier une expression avec termes canoniques. Le théorème stipule que 2 termes canoniques sont adjacents logiques s'ils ne varient que d'une seule variable (directe dans un terme, complémentée dans l'autre). Cette variable est alors superflue dans les 2 termes.

Exemple : $F = f(A,B,C) = \Sigma m(0,3,4,5,7)$

$F = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$

$ABC + ABC$ (théo. 5)

$\bar{B}\bar{C}$ BC AC

$F = \bar{B}\bar{C} + BC + AC$

En fait, pour la simplification d'expressions canoniques (SOP ou POS), l'utilisation du théorème de l'adjacent logique suffit mais elle comporte certaines difficultés : il faut savoir sur quels termes commencer...

Exemple : $F = f(A,B,C)$

$F = \Sigma m(0,1,4,6)$

$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$

$F = \bar{B}\bar{C} + \bar{A}\bar{B}C + AB\bar{C}$

On ne peut plus appliquer le théorème de l'adjacence logique et pourtant, on peut obtenir meilleur résultat toujours avec ce même théorème seul.

$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$

$\bar{A}\bar{B}$ $A\bar{C}$

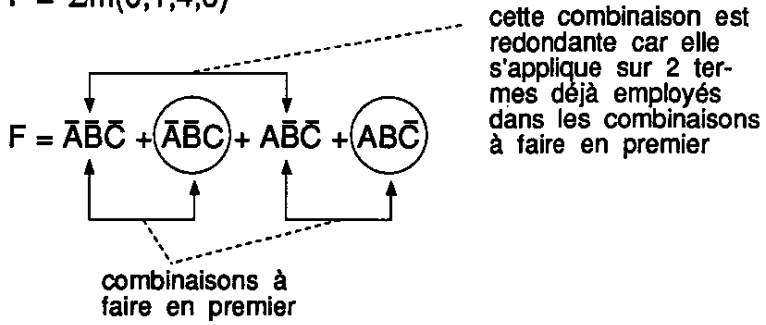
$F = \bar{A}\bar{B} + A\bar{C}$

Règle d'application de l'adjacence logique :

- on doit d'abord regarder toutes les combinaisons possibles de l'application de l'adjacence logique.
- on extrait en premier les combinaisons, dans lesquelles se retrouve un terme qui se combine d'une seule façon.
- on extrait ensuite les autres combinaisons, s'il en reste, en prêtant une attention spéciale aux redondances.
- on recommence les étapes si nécessaire.

Exemple : $F = f(A,B,C)$

$F = \Sigma m(0,1,4,6)$

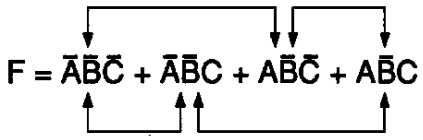


$F = \bar{A}\bar{B} + A\bar{C} + \bar{B}\bar{C}$

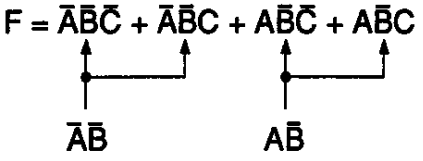
redondant

$F = f(A,B,C)$

$F = \Sigma m(0,1,4,5)$



Tous les termes embarquent dans 2 combinaisons. À ce moment, prendre les combinaisons disjointes et le résultat est tel qu'on peut réappliquer le théorème sur celui-ci :



$F = \bar{B}$

On obtient le même résultat en prenant les 2 autres combinaisons. C'est un chainage du théorème de l'adjacence logique.

3.6 - Méthode de Karnaugh

Développée par Karnaugh (Map Method for Synthesis of Combinational Logic Circuit, 1953), cette méthode permet la simplification d'expressions canoniques (possible pour non-canoniques) basée sur le théorème de l'adjacence logique. C'est une méthode semi-graphique. Le résultat est de la forme SOP (la forme POS est possible mais plus compliquée).

Une fois l'expression de la fonction obtenue, la simplification par la méthode algébrique comporte une difficulté : celle de déterminer systématiquement les termes (monômes canoniques ou minterms en SOP) adjacents de ceux qui ne diffèrent que d'une variable complémentée. La méthode de Karnaugh met en évidence les minterms adjacents et permet d'obtenir facilement une expression logique minimale à 2 couches.

Karnaugh a donc pensé à organiser une carte dans laquelle chaque case se voit assigner un minterm. L'arrangement des cases est tel que chaque case est dans une position géographique adjacente aux cases représentant les minterms en adjacence logique.

Pour une table de vérité à 2 variables

minterm	adjacent à
0	1,2
1	0,3
2	0,3
3	1,2

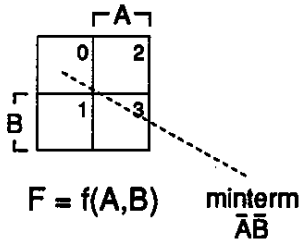
Pour une table de vérité à 3 variables

minterm	adjacent à
0	1,2,4
1	0,3,5
2	0,3,6
3	1,2,7
4	0,5,6
5	1,4,7
6	2,4,7
7	3,5,6

ordre du minterm (m_0)

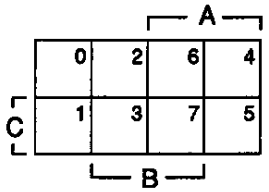
La carte de karnaugh apparait donc comme un rectangle divisé en 2^n petites cases où "n" est le nombre de variables indépendantes qui doit être inférieur ou égal à 4. Pour n = 5, on utilise 2 cartes de 4 variables ; pour n = 6, 4 cartes de 4 variables; etc... Chaque variable apparait aux bords de la carte et les petits numéros qui sont indiqués dans le coin supérieur droit de chaque case, font correspondre l'ordre du minterm associé à la case.

Carte de Karnaugh à 2 v.i.



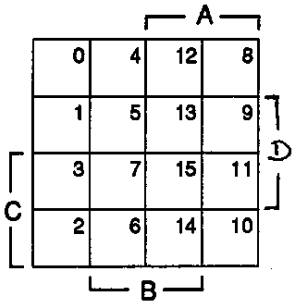
F = f(A,B) minterm $\bar{A}\bar{B}$

Carte de Karnaugh à 3 v.i.



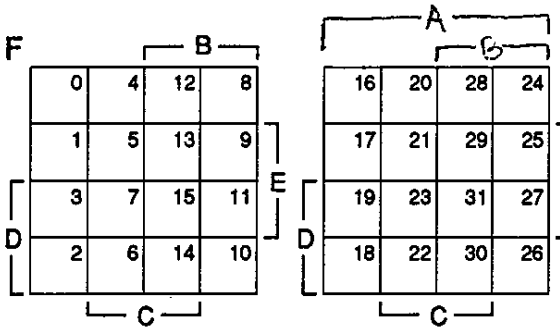
F = f(A,B,C)

Carte de Karnaugh à 4 v.i.



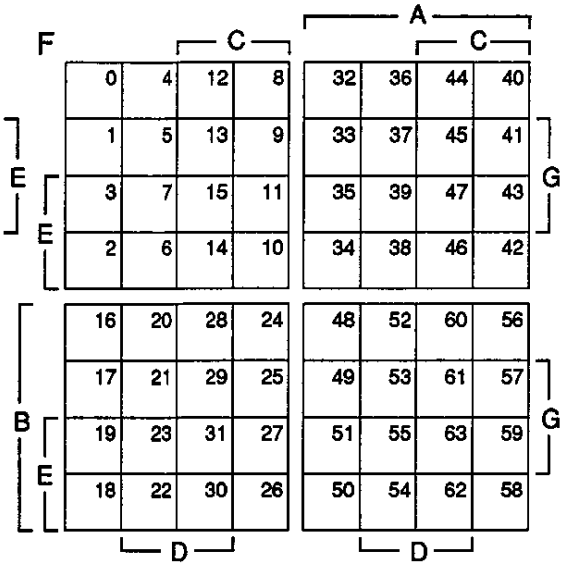
F = f(A,B,C,D)

Carte de Karnaugh à 5 v.i.



F = f(A,B,C,D,E)

Carte de Karnaugh à 6 v.i.



F = f(A,B,C,D,E,G)

Ainsi, les termes qui peuvent se simplifier avec la seule loi de l'adjacence logique sont représentés par des cases voisines sur la carte. Par exemple, avec 3 v.i., le minterm $\bar{A}B\bar{C}$ (m_2) peut se simplifier avec le minterm $\bar{A}\bar{B}\bar{C}$ (m_0) pour donner $\bar{A}\bar{C}$; avec le minterm $\bar{A}BC$ (m_3) pour donner $\bar{A}B$; avec le minterm $AB\bar{C}$ (m_6) pour donner $B\bar{C}$.

m_2 adjacent à m_0, m_3 et m_6

En réalité, le nombre de minterms adjacent à un minterm donné est égal au nombre de variables indépendantes. Or, en regardant la carte, on s'aperçoit que certaines cases des extrémités ne sont pas voisines d'autant de cases que d'autres plus internes. C'est qu'il faut comprendre que chaque carte est repliée sur elle même telle sorte que la ligne horizontale supérieure est adjacente à celle inférieure; que la ligne verticale de gauche est aussi adjacente à celle de droite. Conséquemment, pour la carte de Karnaugh à 4 v.i., le mintem d'ordre 2 (m_2) est voisin des minterms d'ordre 0, 3, 6 et 10 (m_0, m_2, m_6, m_{10}).

$$m_2 = \bar{A}\bar{B}C\bar{D} \begin{cases} \bar{A}\bar{B}\bar{C}\bar{D} = m_0 \\ \bar{A}\bar{B}CD = m_3 \\ \bar{A}BC\bar{D} = m_6 \\ A\bar{B}C\bar{D} = m_{10} \end{cases}$$

Pour utiliser une carte Karnaugh, il est essentiel de commencer par la remplir: dans chaque case, à laquelle est associée un minterm, on indique la condition d'apparition du minterm associé dans la fonction exprimée sous forme SOP.

Exemple : $F = f(A,B,C)$

$F = \bar{A}\bar{B}C + \bar{A}BC + AB\bar{C} + ABC$

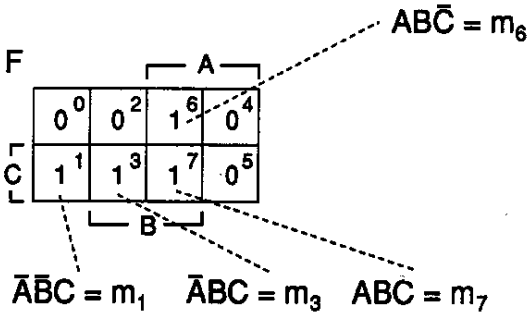
donc

$F = (\bar{A}\bar{B}\bar{C})(0) + (\bar{A}\bar{B}C)(1) + (\bar{A}B\bar{C})(0) + (\bar{A}BC)(1)$
$+ (A\bar{B}\bar{C})(0) + (A\bar{B}C)(0) + (AB\bar{C})(1) + (ABC)(1)$

$+ (ABC)(0) + (ABC)(0) + (ABC)(1) + (ABC)(1)$

condition d'apparition du minterm dans l'expression de la fonction

1 → apparait
0 → n'apparait pas



La seconde étape de l'utilisation d'une carte de Karnaugh est la réduction de la carte. Évidemment, c'est à cette étape que ce fait la réduction de l'expression de la fonction. Pour ce faire, il suffit de former des groupements de cases selon certains critères. Rappelons-nous que chacune des cases équivaut à un minterm. Si chaque case est prise séparément, toutes les variables demeurent, il n'y a aucune simplification. Par contre le groupement de 2 cases voisines permet d'appliquer le théorème de l'adjacence logique ce qui élimine une variable en plus de réduire de 1 le nombre de termes à sommer. Un groupement de 4 cases est possible si on applique le chainage du théorème. Ici, on simplifie 2 variables en plus de passer de 4 termes à un seul terme dans la somme.

Règle de groupement des "1"

$n = 0,1,2,3,4...$

* Il y a toujours 2^n cases dans un groupement (n entier)

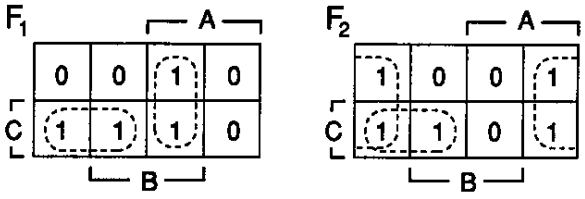
Règle de groupement des "1" (suite)

* Il faut essayer de
  - minimiser le nombre de groupements pour couvrir la carte
  - maximiser la grosseur de chaque groupement

* On peut utiliser des cases déjà couvertes par d'autres groupements pour former un plus gros groupement. Ceci correspond à répéter un minterm plusieurs fois pour aider à simplifier d'avantage.

* Il est inutile de former des groupements n'ayant que des cases déjà couvertes: ceci augmente inutilement la complexité du circuit (attention: dans certains cas, des termes redondants éliminent des aléas des circuits combinatoires. Ce sujet sera dailleurs traité plus loin).

Exemple :



Étapes du processus formel de la réduction de la carte.

a) Grouper d'abord les cases qui ne peuvent être groupées avec aucune autre. Elles forment des ilots.

b) Grouper les cases adjacentes avec lesquelles on ne peut grouper qu'une seule autre case et ce d'une seule façon possible.

c) Répéter l'étape "b" pour des groupements de 4, 8, 16... cases qui se font d'une seule façon possible.

d) Repérer l'ensemble des cases non groupées restantes. Ces cases pouvaient se grouper de plusieurs façons. Faire les groupements en commençant par les plus isolées, ie celles avec les moins de possibilités, et essayer de minimiser le nombre de groupements.

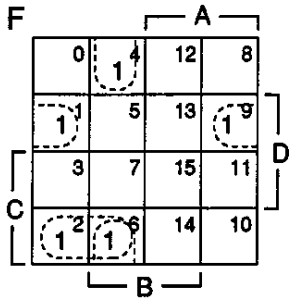
Exemple : soit la fonction

$F = f(A,B,C,D)$

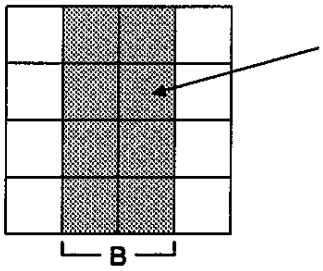
$F = \Sigma m\,(1,2,4,6,9)$

donc

$F = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D$



$m_2 + m_6 = \bar{A}C\bar{D}$ (élimination de B : le groupement étant de part et d'autre de la zone B = 1)



vérification $m_2 + m_6 = \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}$

$\bar{A}C\bar{D}$

$m_1 + m_9 = \bar{B}\bar{C}D$

$m_4 + m_6 = \bar{A}B\bar{D}$

$F = \bar{A}C\bar{D} + \bar{B}\bar{C}D + \bar{A}B\bar{D}$

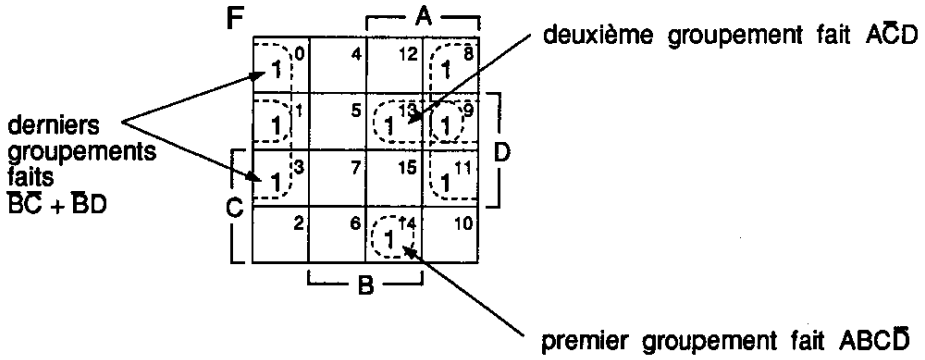
La lecture de la carte constitue la dernière étape de la simplification avec la méthode de Karnaugh. La lecture s'effectue en repérant les zones dans lesquelles se trouve le groupement. Si le groupement se trouve d'une part et d'autre d'une zone, la variable indépendante correspondante est éliminée. Dans l'exemple précédent le groupement ($m_2 + m_6$) est à l'intérieur de la zone (A = 0) ; à l'intérieur de la zone (C = 1); à l'intérieur de la zone (D = 0) mais de part et d'autre de la zone (B = 1) ou (B = 0). (i.e. le regroupement TRAVERSE la frontière entre une variable et son complément)

$m_2 + m_6 \rightarrow$ intérieur zone A = 0, C = 1, D = 0 $\rightarrow \bar{A}C\bar{D}$

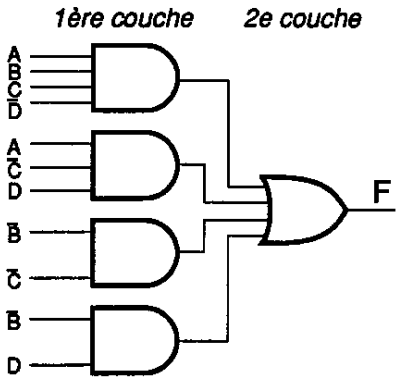
$\rightarrow$ v.i. B éliminée

Exemple #1 : $F = f(A,B,C,D)$

$F = \Sigma m(0,1,3,8,9,11,13,14)$



$F = ABC\bar{D} + A\bar{C}D + \bar{B}\bar{C} + \bar{B}D$



L'emploi des tables de Karnaugh pour simplifier une expression du type SOP donne une expression à 2 couches dite SSOP. Cette expression SSOP est la plus simple obtenue à 2 couches lorsqu'on utilise les tables de Karnaugh rigoureusement.*

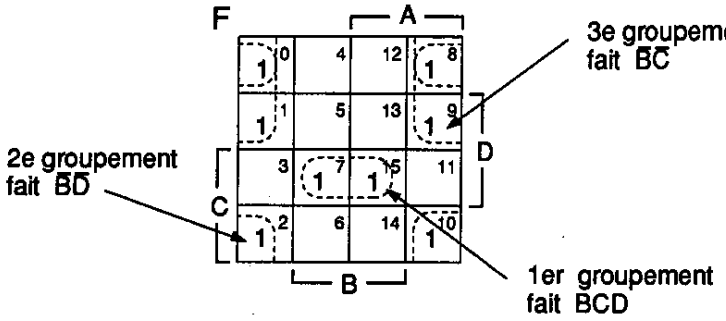
* *SSOP : "Simplified Sum Of Products"*

Notes : *Une réalisation à 2 couches implique le passage par 2 portes pour tous les signaux. (une expression SSOP nécessite un étage de "AND" et un étage de "OR"). Évidemment, il est possible de simplifier d'avantage une expression "SSOP" obtenue par l'emploi des tables de Karnaugh, mais cette simplification conduira vers une réalisation multi-couches.*

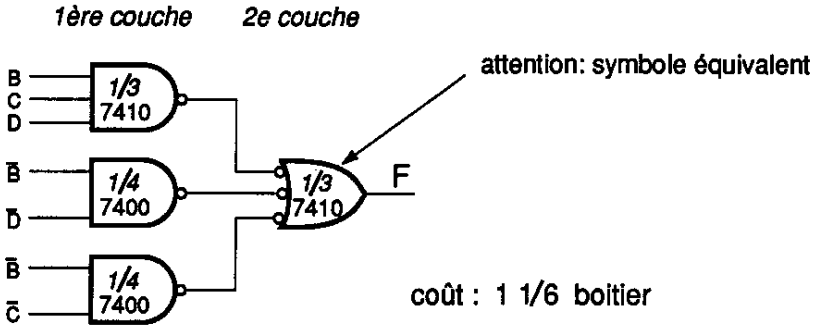
La réalisation à 2 couches est la réalisation la plus rapide car il existe un délai de propagation du signal franchissant une porte. Le nombre de passage par des portes doit donc être minimisé et ce nombre minimum pour une expression moindrement complexe est 2.

Exemple #2 : $F = f(A,B,C,D)$

$F = \Sigma m(0,1,2,7,8,9,10,15)$



$F = BCD + \bar{B}\bar{D} + \bar{B}\bar{C}$



Le coût (en boitiers) du circuit s'établit de la façon suivante :

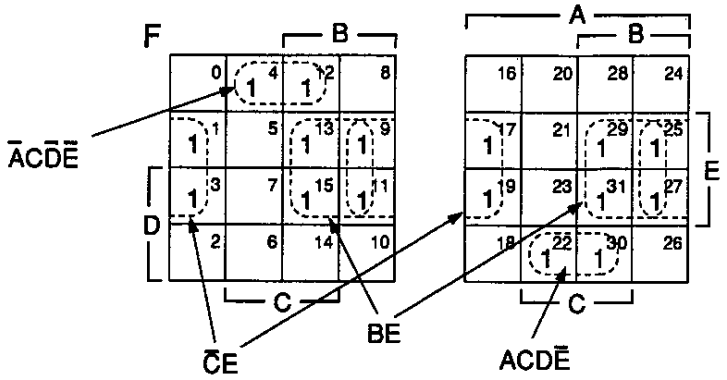
- *Il y a 3 portes "NAND" à 3 entrées dans un 7410. Chaque porte équivaut donc à 1/3 de boitier*
- *Il y a 4 portes "NAND" à 2 entrées dans un 7400. Chaque porte équivaut donc à 1/4 de boitier*

CONVENTION → Les entrées peuvent être accessibles sous forme directe ou complémentée: on ne compte donc pas les inverseurs nécessaires aux entrées des v.ind. dans un circuit combinatoire. Il y va de même pour la sortie qui peut être fournie sous forme directe ou complémentée.

	2/3	7410
+	1/2	7400
	1 1/6	boitiers

Exemple #3 : $F = f(A,B,C,D,E)$

$F = \Sigma m(1,3,4,9,11,12,13,15,17,19,22,25,27,29,30,31))$



$F = \bar{A}C\bar{D}\bar{E} + ACD\bar{E} + \bar{C}E + BE$

il faut voir les 2 cartes de 4 variables comme étant superposées.

Par exemple m_1 ($\bar{A}\bar{B}\bar{C}\bar{D}E$) est adjacent à m_{17} ($A\bar{B}\bar{C}\bar{D}E$) puisque seul la variable "A" change.

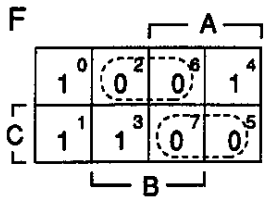
Coût : 1 1/2 7420 (3 portes "NAND" à 4 entrées)
\+ 1/2 7400 (2 portes "NAND" à 2 entrées)
2 boitiers

Jusqu'à présent, on s'est servi des tables de Karnaugh pour réaliser la fonction "F" sous forme directe. Mais, sous certaines conditions qui seront vues plus tard, il vaut mieux obtenir "$\bar{F}$". L'obtention de l'expression en vue d'obtenir "$\bar{F}$" se fait simplement en regroupant les "0" dans la table plutôt que les "1".

regroupement des "1" ⟶ F
regroupement des "0" ⟶ $\bar{F}$

Exemple #4 : $F = f(A,B,C)$

$F = \Sigma m(0,1,3,4)$



$\bar{F} = B\bar{C} + AC$

Vérification :

$F = \bar{A}C + B\bar{C}$

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

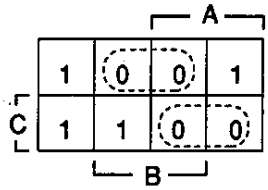
$\bar{F} = (\overline{\bar{A}C}) \cdot (\overline{B\bar{C}})$

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

$\bar{F} = AB + AC + B\bar{C} + \underbrace{C\bar{C}}_{0 \text{ (th.7)}}$

terme redondant car $AC = \Sigma m(5,7)$, $BC = \Sigma m(2,6)$, $AB = \Sigma m(6,7)$

$\bar{F} = AC + B\bar{C}$



3.7 - Synthèse pour la forme "SPOS"

Lorsque l'expression de la fonction est décrite avec les maxterms, il demeure possible de garder la forme d'un produit de somme après la simplification et ce, toujours en se servant des tables de Karnaugh.

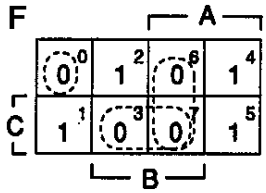
L'expression simplifiée de cette forme (SPOS) est la forme duale de celle obtenue avec les minterms et l'usage normal des tables de Karnaugh. Elle s'implante avec des "NOR" seulement à deux couches.

Les étapes sont :

* *Remplir la table de Karnaugh en entrant un "0" plutôt qu'un "1" dans les cases dont le numéro correspond à l'ordre de l'un des maxterms de l'équation. Entrer un "1" dans le cas contraire. (Il ne faut pas oublier que chaque case est associée à un minterm).*
* *Faire la synthèse par les "0" pour sortir l'expression de "$\bar{F}$" sous la forme d'une somme de produit. (SSOP).*
* *Appliquer les lois de De Morgan pour transposer l'équation dans l'autre forme (SPOS).*

Exemple : $F = f(A,B,C)$

$F = \Pi M(0,3,6,7)$



$F = \Sigma m(1,2,4,5)$

$\bar{F} = \bar{A}\bar{B}\bar{C} + AB + BC$

$\bar{\bar{F}} = F = \overline{\bar{A}\bar{B}\bar{C} + AB + BC}$

$= \overline{\bar{A}\bar{B}\bar{C}} \cdot \overline{AB} \cdot \overline{BC}$

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

3.8 - Les valeurs indifférentes

Souvent, dans un système numérique, certains codes d'entrée sont supposés ne jamais arriver ou se produire ou encore ne pas avoir d'effet sur le résultat. On dit alors que pour ces codes, le résultat est indifférent.

Un résultat indifférent d'une fonction, dénoté par un X dans la table de vérité et de Karnaugh, est une valeur non spécifiée. Elle peut être un "1" ou un "0".

Résultat indifférent ⟶ X

Notation utilisée pour l'expression d'une fonction avec résultat indifférent

$F = f(A,B,C) = \Sigma m(0,2,7) + x\,(3,4)$

le résultat est indifférent pour les combinaisons des entrées auxquelles correspondent les minterms d'ordre 3 et 4.

Un exemple typique de l'usage des valeurs indifférentes apparait lorsqu'on utilise le code "BCD" ("Binary Coded Decimal"), qui traduit une valeur décimale de 0 à 9 sur un code binaire de 4 bits habituellement. Comme le code binaire représenté sur 4 bits s'étend de 0 à 15, il se trouve que les combinaisons 10 à 15 ne sont pas utilisées et que, par conséquent, on peut attribuer des valeurs favorisant la simplification du circuit aux valeurs de la fonction pour ces combinaisons.

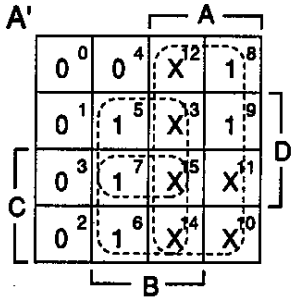
Exemple : Il existe plusieurs formes de codage BCD : Le plus répendu étant le code BCD 8421 (appelé plus simplement BCD) où les 4 chiffres donnent le poids de chaque bit du code. Ainsi, 1 0 0 1 représente 9 puisque :

1 x 8 = 8
0 x 4 = 0
0 x 2 = 0
1 x 1 = 1
9

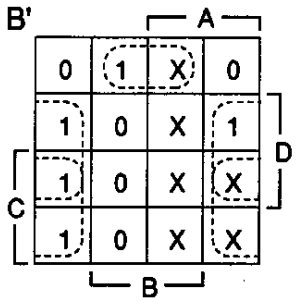
Réalisons maintenant un transcodeur faisant la conversion entre un code BCD 8421 et un code BCD 84-2-1

4 fonctions des mêmes v.ind.

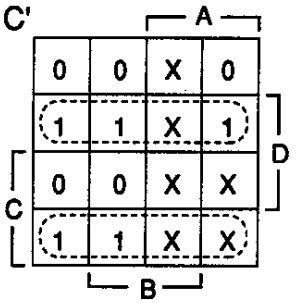
Valeur décimale	A	B	C	D	A'	B'	C'	D'	
0	0	0	0	0	0	0	0	0	
1	0	0	0	1	0	1	1	1	
2	0	0	1	0	0	1	1	0	
3	0	0	1	1	0	1	0	1	
4	0	1	0	0	0	1	0	0	
5	0	1	0	1	1	0	1	1	axe de symétrie
6	0	1	1	0	1	0	1	0	
7	0	1	1	1	1	0	0	1	
8	1	0	0	0	1	0	0	0	
9	1	0	0	1	1	1	1	1	
10	1	0	1	0	X	X	X	X	les codes BCD de
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	valeur supérieure à
	1	1	1	1	X	X	X	X	9 ne peuvent arriver



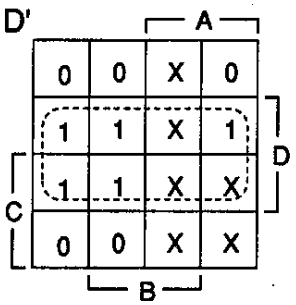
$A' = A + BC + BD$



$B' = B\bar{C}\bar{D} + \bar{B}C + \bar{B}D$



$C' = \bar{C}D + C\bar{D}$



$D' = D$

L'exemple précédent illustre très bien l'avantage d'avoir le plus grand nombre de combinaisons auxquels correspondent des valeurs indifférentes de la sortie. On se sert d'une valeur indifférente dans le seul cas ou le groupement formé résulte en meilleure simplification. Les "X" encerclés prennent la valeur "1" et les autres doivent être vus comme des "0".

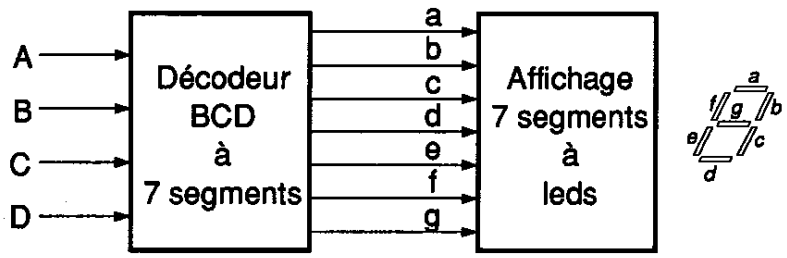
3.9 - Les multifonctions

Le dernier exemple traité est un exemple de multifonction ou de fonctions simultanées. Ce circuit à 4 sorties (A',B',C',D'), fut traité comme 4 problèmes indépendants où chacune des fonctions dépendait des 4 mêmes variables indépendantes (A,B,C,D). On avait donc :

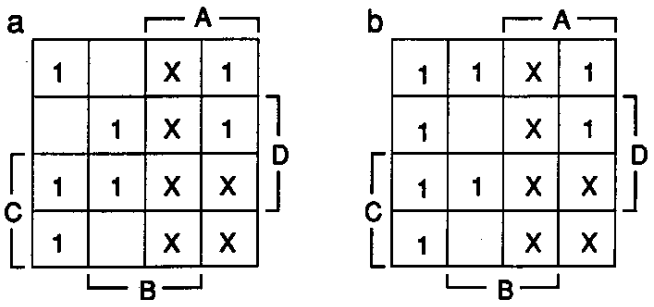
A'(A,B,C,D)
B'(A,B,C,D)
C'(A,B,C,D)
D'(A,B,C,D)

Il est cependant possible de réaliser des économies supplémentaires en matériel, en fusionnant ces fonctions dans le but d'utiliser des groupements de l'une des fonctions dans la réalisation d'une autre.

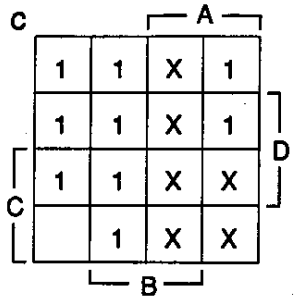
Exemple : Décodeur "BCD" à 7 segments :



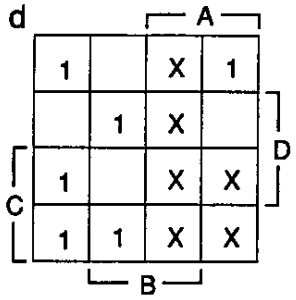
Valeur décimale	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1



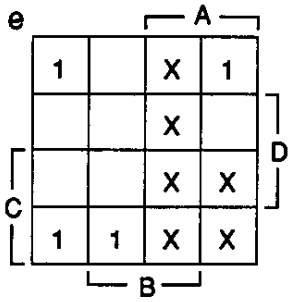
c



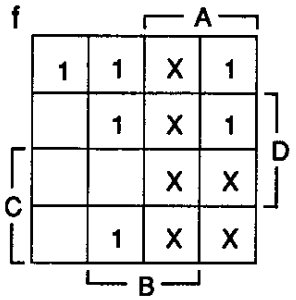
d



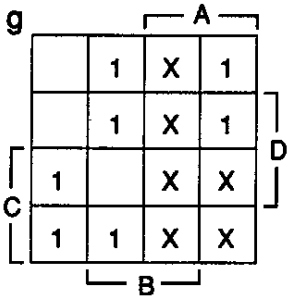
e



f



g



Il faut essayer de trouver des groupements qui risquent de servir à plusieurs fonctions à la fois pour minimiser le coût en boitiers. Par exemple, le terme $B\bar{C}D$ peut servir dans les fonctions a, d, f et g.

$$a = B\bar{C}D + CD + \bar{B}\bar{D} + A$$

$$b = CD + \bar{C}\bar{D} + \bar{B}$$

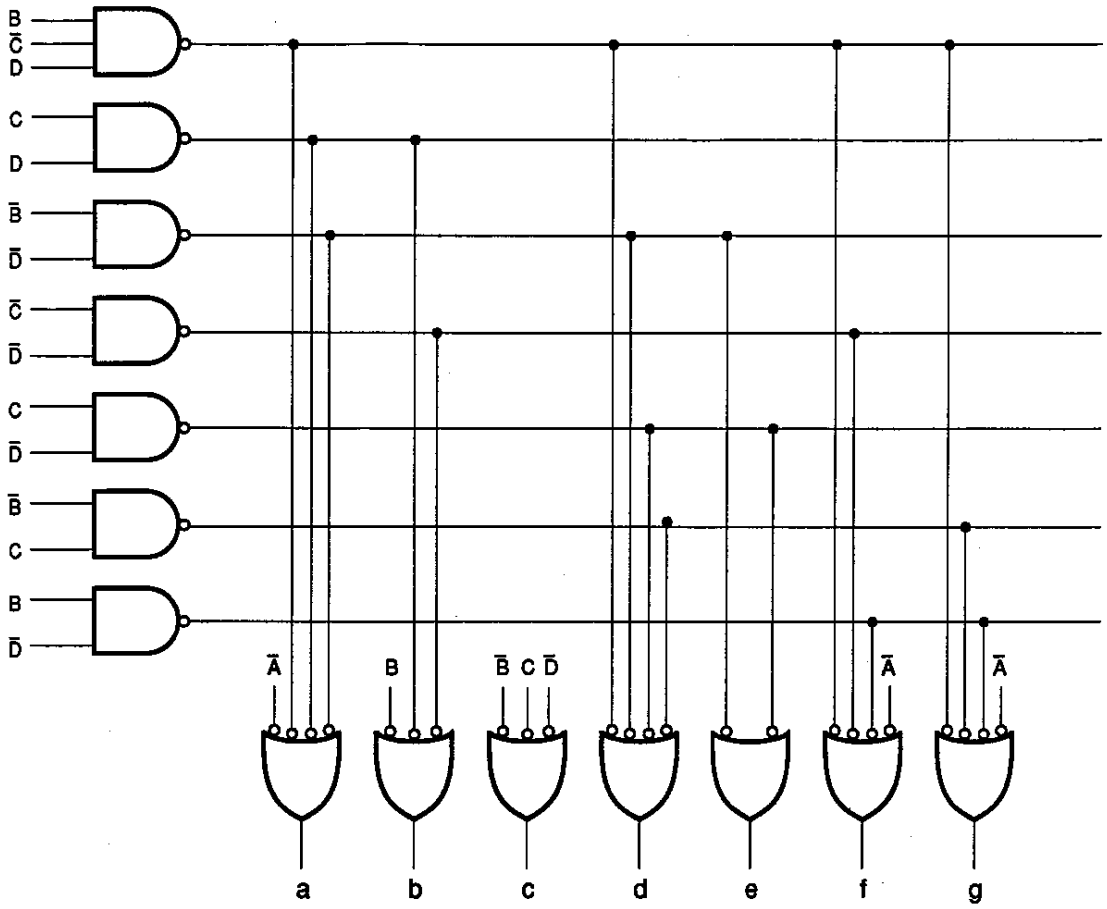
$$c = B + \bar{C} + D$$

$$d = B\bar{C}D + \bar{B}\bar{D} + C\bar{D} + \bar{B}C$$

$$e = \bar{B}\bar{D} + C\bar{D}$$

$$f = B\bar{C}D + \bar{C}\bar{D} + B\bar{D} + A$$

$$g = B\bar{C}D + \bar{B}C + B\bar{D} + A$$



coût :

2	7420
1	7410
1 3/4	7400
4 3/4	boitiers

Notez que le TTL 7447 est aussi un décodeur BCD - 7 segments (donc coût de 1 boitier) mais "A" est la bit la moins significative... attention au poids des bits (habituellement "A" à poids faible en TTL).

3.9.1 - Design de multi-fonctions par possibilités.

Soit trois fonctions logiques F_1, F_2, F_3, toutes fonctions des mêmes variables indépendantes A, B, C. Les tables de vérité des 3 fonctions sont :

A	B	C	F_1	F_2	F_3
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	0	1	0

En solutionnant les tables ci-dessous sans tenir compte des multi-fonctions on a :

$F_1 = A\bar{B}C + \bar{A}B + B\bar{C} + \bar{A}\bar{C}$

$F_2 = A\bar{B}\bar{C} + \bar{A}B + \bar{A}C + BC$

$F_3 = A\bar{B} + \bar{A}B + B\bar{C}$

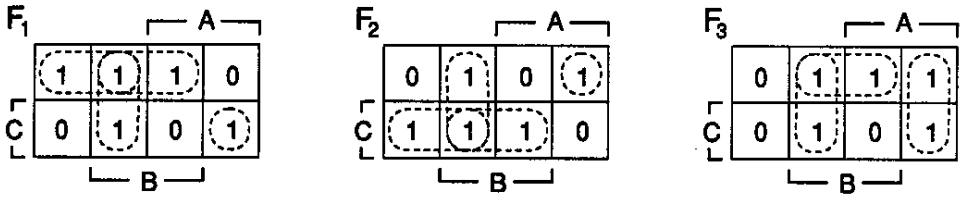
coûts : $F_1 = 1/3 + 1/4 + 1/4 + 1/4 + 1/2 = 1\ 7/12$

$F_2 = 1/3 + 1/4 + 1/4 + 1/4 + 1/2 = 1\ 7/12$

$F_3 = 1/4 + 1/4 + 1/4 + 1/3 = 1\ 1/12$

$4\ 3/12$

Les tables de Karnaugh sont :



La table de tous les groupements possibles pour chaque minterm des fonctions F_1 , F_2 et F_3 est la suivante :

MINTERM	F_1	F_2	F_3
m_0	$\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{C}$	—	—
m_1	—	$\bar{A}\bar{B}C$, $\bar{A}C$	—
m_2	$\bar{A}B\bar{C}$, $\bar{A}\bar{C}$, $\bar{A}B$, $B\bar{C}$	$\bar{A}B\bar{C}$, $\bar{A}B$	$\bar{A}B\bar{C}$, $\bar{A}B$, $B\bar{C}$
m_3	$\bar{A}BC$, $\bar{A}B$	$\bar{A}BC$, $\bar{A}B$, $\bar{A}C$, BC	$\bar{A}BC$, $\bar{A}B$
m_4	—	$A\bar{B}\bar{C}$	$A\bar{B}\bar{C}$, $A\bar{C}$, $A\bar{B}$
m_5	$A\bar{B}C$	—	$A\bar{B}C$, $A\bar{B}$
m_6	$AB\bar{C}$, $B\bar{C}$	—	$AB\bar{C}$, $B\bar{C}$
m_7	—	ABC, BC	—

On regarde dans cette table, les groupements communs aux fonctions. On procède comme suit :

1) *On écrit les termes des lignes pour lesquels aucun groupement n'est commun*
2) *On traite ensuite les lignes avec 2 éléments communs*
3) *On poursuit avec les lignes avec 3 éléments communs*
4) *On prend en donnant dans chaque cas les plus petits termes communs*

$F_1 = \bar{A}\bar{C} + (\ ) + (\ ) + \bar{A}B$ coûts : $1/4 + 0 + 0 + 1/4 + 1/2 = 1$

$F_2 = BC + \bar{A}C + A\bar{B}\bar{C} + (\ )$ coûts : $1/4 + 1/4 + 1/3 + 0 + 1/2 = 1\ 4/12$

$F_3 = (\ ) + A\bar{B}C + B\bar{C} + (\ )$ coûts : $0 + 1/3 + 1/4 + 0 + 1/2 = 1\ 1/12$

$3\ 5/12$

Donc on ne gagne pas grand chose dans ce cas!

Regarder selon les colonnes (comparer chaque fonction avec ou sans cette possibilité de multifonctions) et prendre la meilleure solution.

3.9.2 - Design de multifonctions par la méthode des indifférents

L'exemple ici-bas donne une approche systématique d'utilisation d'une fonction F (A,B,C) pour générer Z (A,B,C) en utilisant le concept des états indifférents. Connaissant les 2 fonctions, on construit une table de vérité de Z (A,B,C, F (A,B,C)) dans laquelle on remarque qu'au moins la moitié des combinaisons de A,B,C et F (A,B,C) sont impossibles (donc indifférentes) car F(A,B,C) ne peut dans ces cas donner la valeur spécifiée par la table. On résoud alors la nouvelle table de Karnaugh pour Z normalement.

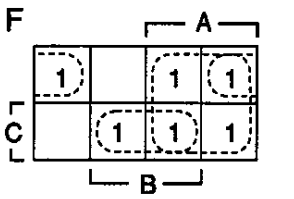
Exemple :

F(A,B,C)

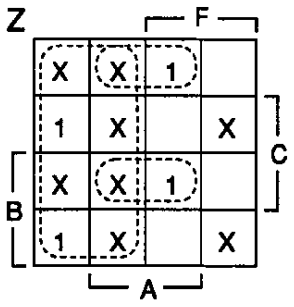
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Z(A,B,C)

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$F = \bar{B}\bar{C} + BC + A$



$Z = A\bar{B}\bar{C} + ABC + \bar{F}$

(voir table de vérité à la page suivante)

Le circuit, ainsi synthétisé, n'est plus à 2 couches contrairement à la technique précédente qui offre toujours un circuit à 2 couches.

F	A	B	C	Z
0	0	0	0	X
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	X
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	X
1	0	1	0	X
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

← condition impossible puisque F(A,B,C) = 1 avec A = B = C = 0

3.10 - Lecture des "EXOR" avec Karnaugh

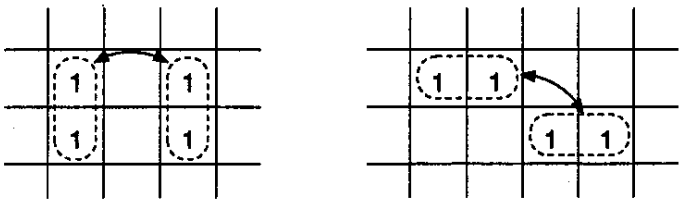
Certaines techniques simples permettent de reconnaître l'utilisation d'une porte spéciale, "EXOR" (ou exclusif), dans une table de Karnaugh. Cependant, l'expression obtenue ne fait plus partie des familles "SSOP" ou "SPOS" à 2 couches. Il va sans dire que l'emploi de ces portes implique un nombre de couches sans importance et généralement une meilleure réduction du nombre de boitiers particulièrement dans les circuits arithmétiques.

Il existe 2 agencements de groupements qui indiquent la possibilité d'utilisation d'une porte "EXOR" dans une table de Karnaugh : l'adjacence avec "Offset" et l'adjacence avec "Kitty corner".

agencement de groupements pour utilisation de EXOR → "Offset"
→ "Kitty corner"

"Offset" : *Cet adjacence permet d'unir des groupements qui sont décalés d'une case verticalement ou horizontalement dans une table de Karnaugh.*

"Kitty corner": *Cet adjacence permet d'unir des groupements qui ne sont pas en adjacence logique mais qui se touchent par un de leurs coins.*



"Offset" "Kitty corner":

La lecture se fait de la manière suivante :

* *Repérer les groupements susceptibles de s'unir par un*
  EXOR ⟶ "Offset"
  ⟶ "Kitty corner"

* *Les groupements doivent se situer de part et d'autre de 2 zones de v.i dans la table de Karnaugh. Ces 2 v.i associées font partie de l'expression du EXOR*

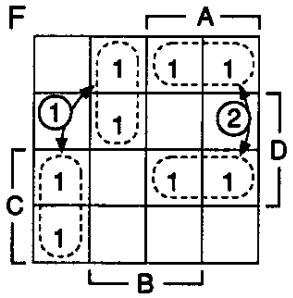
* *Déterminer s'il s'agit d'un EXOR ou d'un NEXOR. Pour ce faire, il faut prendre l'un des groupements et regarder s'il se trouve à l'intérieur de ces 2 zones. Dans ces cas, il s'agit d'un "NEXOR" ; autrement, il s'agit d'un "EXOR".*

  Intérieur ou extérieur } des 2 zones ⟶ NEXOR

  à l'intérieur de l'une à l'extérieur de l'autre } ⟶ EXOR

* *Déterminer le facteur de multiplication en regardant les variables communes à l'espace occupé par les 2 groupements.*

Exemple :



① les 2 groupements sont "Kitty corner"

② les 2 groupements sont en "Offset"

- Les 2 groupements de ① sont de part et d'autre des zones des v.i B et C
- Les 2 groupements de ② sont de part et d'autre des zones des v.i C et D
- Le groupement inférieur de ① est à l'extérieur de la zone (B=1) mais à l'intérieur de la zone (C=1). C'est donc un EXOR
- Le groupement inférieur de ② est à l'intérieur de la zone (C=1) et de la zone (D=1). C'est un NEXOR
- Les 2 groupements de ① sont dans la zone " $\bar{A}$" tandis que les 2 groupements de ② sont dans "A"

$\therefore$ $F = \bar{A}(B \oplus C) + A(C \odot D)$

$F = \bar{A}(B \oplus C) + A(\overline{C \oplus D})$

$F = \bar{A}(B \oplus C) + A(C \oplus \bar{D})$

3.11 - Variables conditionnelles (VEM)

Les tables de Karnaugh sont limitées à 6 variables. Dans les systèmes où il y a plus de 6 variables indépendantes, il faut trouver un autre moyen graphique de solution des problèmes de logique combinatoire. En particulier, Dans les systèmes séquentiels synchrones à plusieurs états, le nombre de variables indépendantes devient très grand et il devient utile de faire la différence entre les variables d'état et les variables externes (voir chap. 7).

Les variables conditionnelles (ou "VEM": "Variable Entered Mapping") permettent de généraliser la technique des tables de Karnaugh à un nombre de variables indépendantes supérieur à 6. Le principe des "VEM" consiste à ***placer les variables indépendantes dans les cases de la table de Karnaugh.*** *Une technique spéciale de lecture des tables avec "VEM" permet d'obtenir l'expression logique de la fonction.*

tables avec "VEM" ⟶ 0, 1, X et variables inépendantes dans la table

3.11.1 - Justification théorique du principe des variables conditionnelles

Soit une fonction : F = f(A,B,C)

A	B	C	F
0	0	0	f_0
0	0	1	f_1
0	1	0	f_2
0	1	1	f_3
1	0	0	f_4
1	0	1	f_5
1	1	0	f_6
1	1	1	f_7

On peut écrire la somme de produits :

$$F = m_0f_0 + m_1f_1 + m_2f_2 + m_3f_3 + m_4f_4 + m_5f_5 + m_6f_6 + m_7f_7$$

où : $f_i = 0$, si le minterm n'apparaît pas dans F
$= 1$, si le minterm apparaît dans F
$= X$, si le minterm est indifférent

On peut écrire F sous la forme :

$$F = \bar{A}\bar{B}\bar{C}f_0 + \bar{A}\bar{B}Cf_1 + \bar{A}B\bar{C}f_2 + \bar{A}BCf_3 + A\bar{B}\bar{C}f_4 + A\bar{B}Cf_5 + AB\bar{C}f_6 + ABCf_7$$

En utilisant les théorèmes de l'algèbre de Boole, on peut factoriser :

$$F = \bar{A}\bar{B}(\bar{C}f_0 + Cf_1) + \bar{A}B(\bar{C}f_2 + Cf_3) + A\bar{B}(\bar{C}f_4 + Cf_5) + AB(\bar{C}f_6 + Cf_7)$$

en posant : $\bar{C}f_0 + Cf_1 = f'_0(\bar{C},C)$, $\bar{A}\bar{B} = m'_0$

$\bar{C}f_2 + Cf_3 = f'_1(\bar{C},C)$, $\bar{A}B = m'_1$

$\bar{C}f_4 + Cf_5 = f'_2(\bar{C},C)$, $A\bar{B} = m'_2$

$\bar{C}f_6 + Cf_7 = f'_3(\bar{C},C)$, $AB = m'_3$

On peut écrire pour F(A,B,C)

$F(A,B,C) = m'_0 f'_0(\bar{C},C) + m'_1 f'_1(\bar{C},C) + m'_2 f'_2(\bar{C},C) + m'_3 f'_3(\bar{C},C)$

Ce qui est encore une expression logique qui est fonction de 3 variables indépendantes A, B, C, mais qui peut être vue comme un problème à 2 variables (A et B) avec une ***variable conditionnelle*** *(ici, lavariable C). La table de Karnaugh prend la forme :*

B \ A		
	0 : $f'_0(\bar{C},C)$	2 : $f'_2(\bar{C},C)$
	1 : $f'_1(\bar{C},C)$	3 : $f'_3(\bar{C},C)$

On voit qu'ici, les cases contiennent des ***fonctions*** *de la variable C au lieu des "0" et des "1" habituels. Les m'_i sont appelés des sub-minterms. Les f_i sont les conditions d'apparition de ces sub-minterms dans F.*

Prenons un exemple concret, soit la table de vérité suivante :

A	B	C	F
0	0	0	$0 = f_0$
0	0	1	$0 = f_1$
0	1	0	$1 = f_2$
0	1	1	$1 = f_3$
1	0	0	$1 = f_4$
1	0	1	$0 = f_5$
1	1	0	$X = f_6$
1	1	1	$X = f_7$

En prenant C comme variable conditionnelle on peut écrire :

$F = \bar{A}\bar{B}(\bar{C}f_0 + Cf_1) + \bar{A}B(\bar{C}f_2 + Cf_3) + A\bar{B}(\bar{C}f_4 + Cf_5) + AB(\bar{C}f_6 + Cf_7)$

$f'_0(\bar{C},C) = \bar{C}f_0 + Cf_1 = \bar{C}\cdot 0 + C0 = 0$

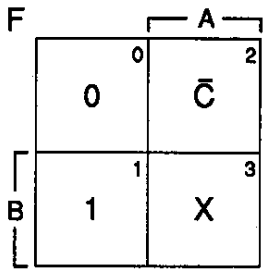
$f'_1(\bar{C},C) = \bar{C}f_2 + Cf_3 = \bar{C}\cdot 1 + C\cdot 1 = \bar{C} + C = 1$

$f'_2(\bar{C},C) = \bar{C}f_4 + Cf_5 = \bar{C}\cdot 1 + C\cdot 0 = \bar{C}$

$f'_3(\bar{C},C) = \bar{C}f_6 + Cf_7 = \bar{C}X + CX = X$

$\longrightarrow F = \bar{A}\bar{B}(0) + \bar{A}B(1) + A\bar{B}(\bar{C}) + AB(X)$

La table de Karnaugh se construit donc comme suit :



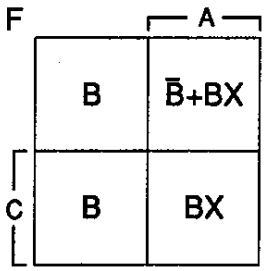
ici, "C" est choisie comme variable conditionnelle.

On aurait pu aussi écrire, en choisissant B comme variable conditionnelle :

$$F = \bar{A}\bar{C}(\bar{B}f_0 + Bf_2) + \bar{A}C(\bar{B}f_1 + Bf_3) + A\bar{C}(\bar{B}f_4 + Bf_6) + AC(\bar{B}f_5 + Bf_7)$$

ici

$$\bar{B}f_0 + Bf_2 = \bar{B}\cdot 0 + B\cdot 1 = B$$
$$\bar{B}f_1 + Bf_3 = \bar{B}\cdot 0 + B\cdot 1 = B$$
$$\bar{B}f_4 + Bf_6 = \bar{B}\cdot 1 + B\cdot X = \bar{B} + BX$$
$$\bar{B}f_5 + Bf_7 = \bar{B}\cdot 0 + B\cdot X = BX$$



On constate donc que l'allure de la table de Karnaugh dépend de la variable conditionnelle choisie. Cependant, l'expression logique pour F ne devrait pas dépendre de la variable conditionnelle choisie.

On peut même choisir plus d'une variable conditionnelle pour réduire les dimensions d'une table de Karnaugh. Prenons l'exemple suivant :

A	B	C	D	F
0	0	0	0	$1 = f_0$
0	0	0	1	$1 = f_1$
0	0	1	0	$1 = f_2$
0	0	1	1	$X = f_3$
0	1	0	0	$1 = f_4$
0	1	0	1	$0 = f_5$
0	1	1	0	$X = f_6$
0	1	1	1	$0 = f_7$
1	0	0	0	$0 = f_8$
1	0	0	1	$1 = f_9$
1	0	1	0	$1 = f_{10}$
1	0	1	1	$1 = f_{11}$
1	1	0	0	$X = f_{12}$
1	1	0	1	$X = f_{13}$
1	1	1	0	$0 = f_{14}$
1	1	1	1	$0 = f_{15}$

Prenons C et D comme variables conditionnelles
On peut écrire :

$$F = \bar{A}\bar{B}\bar{C}\bar{D}f_0 + \bar{A}\bar{B}\bar{C}Df_1 + \bar{A}\bar{B}C\bar{D}f_2 + \bar{A}\bar{B}CDf_3 + \bar{A}B\bar{C}\bar{D}f_4 + \bar{A}B\bar{C}Df_5$$
$$+\bar{A}BC\bar{D}f_6 + \bar{A}BCDf_7 + A\bar{B}\bar{C}\bar{D}f_8 + A\bar{B}\bar{C}Df_9 + A\bar{B}C\bar{D}f_{10} + A\bar{B}CDf_{11}$$
$$+AB\bar{C}\bar{D}f_{12} + AB\bar{C}Df_{13} + ABC\bar{D}f_{14} + ABCDf_{15}$$

en factorisant :

$$F = \bar{A}\bar{B}\left[\bar{C}\bar{D}f_0 + \bar{C}Df_1 + C\bar{D}f_2 + CDf_3\right] +$$
$$\bar{A}B\left[\bar{C}\bar{D}f_4 + \bar{C}Df_5 + C\bar{D}f_6 + CDf_7\right] +$$
$$A\bar{B}\left[\bar{C}\bar{D}f_8 + \bar{C}Df_9 + C\bar{D}f_{10} + CDf_{11}\right] +$$
$$AB\left[\bar{C}\bar{D}f_{12} + \bar{C}Df_{13} + C\bar{D}f_{14} + CDf_{15}\right]$$

En consultant la table de vérité pour avoir la condition d'apparition des sub-minterms, m'_i

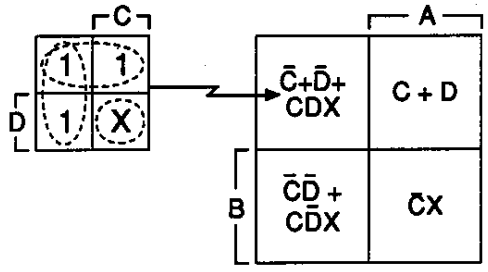
$$\rightarrow \bar{C}\bar{D}f_0 + \bar{C}Df_1 + C\bar{D}f_2 + CDf_3 = \bar{C}\bar{D}\cdot 1 + \bar{C}D\cdot 1 + C\bar{D}\cdot 1 + CD\cdot X$$
$$= \bar{C}\bar{D} + \bar{C}\bar{D} + \bar{C}D + C\bar{D} + CDX$$
$$= \bar{C} + \bar{D} + CDX$$

$$\rightarrow \bar{C}\bar{D}f_4 + \bar{C}Df_5 + C\bar{D}f_6 + CDf_7 = \bar{C}\bar{D}\cdot 1 + \bar{C}D0 + C\bar{D}X + CD\cdot 0$$
$$= \bar{C}\bar{D} + C\bar{D}X$$

$$\rightarrow \bar{C}\bar{D}f_8 + \bar{C}Df_9 + C\bar{D}f_{10} + CDf_{11} = \bar{C}\bar{D}\cdot 0 + \bar{C}D\cdot 1 + C\bar{D}\cdot 1 + CD\cdot 1$$
$$= \bar{C}D + C\bar{D} + CD + CD$$
$$= C + D$$

$$\rightarrow \bar{C}\bar{D}f_{12} + \bar{C}Df_{13} + C\bar{D}f_{14} + CDf_{15} = \bar{C}\bar{D}X + \bar{C}DX + C\bar{D}\cdot 0 + CD\cdot 0$$
$$= \bar{C}\bar{D}X + \bar{C}DX$$
$$= \bar{C}X$$

La table de karnaugh avec 2 variables conditionnelles a la forme suivante :



Le fait d'utiliser des variables conditionnelles permet de réduire les dimensions de la table de Karnaugh de 4 à 2

La convention suivante est adoptée :

$$CX = \begin{cases} 0 \text{ si } C = 0 \\ X \text{ si } C = 1 \end{cases} \qquad \bar{C}X = \begin{cases} X \text{ si } C = 0 \\ 0 \text{ si } C = 1 \end{cases}$$

$$\bar{C} + CX = \begin{cases} 1 \text{ si } C = 0 \\ X \text{ si } C = 1 \end{cases} \qquad \text{etc...}$$

Il faut maintenant définir une technique de lecture des tables avec "VEM" (variables conditionnelle).

3.11.2 - Lecture des tables de Karnaugh avec VEM

La lecture des tables de Karnaugh n'est pas une tâche facile, surtout lorsque l'on retrouve plusieurs "VEM" dans une même case ou lorsqu'une case contient des sommes et/ou produits de VEM.

La procédure à suivre est la suivante :

Étape 1 : Traiter d'abord les VEM dans la table.

- a) *Encercler toutes les VEM simples qui ne peuvent être groupées avec une autre VEM identique et adjacente, un "1" ou un "X".*

  Remarque : C et $\bar{C}$ ne sont pas identiques car elles ne correspondent pas à la même région de la table de Karnaugh sans VEM.

  Remarque : CX, $\bar{C}X$, $C + \bar{C}X$ ne sont pas des VEM simples.

- b) *Encercler ensuite les VEM identiques adjacentes en cherchant les groupements les plus gros possibles.*

- c) *Grouper les cases avec des VEM simples dans une case et des "1" dans les autres.*

  Remarque : Toutes les VEM pouvant se grouper de plusieurs façons avec des VEM identiques et adjacentes, des "1" ou des "X" mais sans faire partie d'un groupement plus gros doivent être groupées en dernier lieu en leur portant une attention particulière.

- d) *Grouper les cases avec des VEM simples dans une case et des "X" dans les autres.*

  → *Chercher les groupements les plus gros.*

  → *Traiter les VEM simples les unes après les autres.*

Étape 2 : Une fois les VEM simples groupées, on transforme la table pour obtenir une table secondaire duquel on pourra retirer les termes de l'expression qui ne contiendront pas une variable conditionnelle. On observe alors les règles suivantes :

- a) *Remplacer les VEM simples par des "0"*

- b) *Remplacer les "0" par des "0"*
  les "X" par des "X"

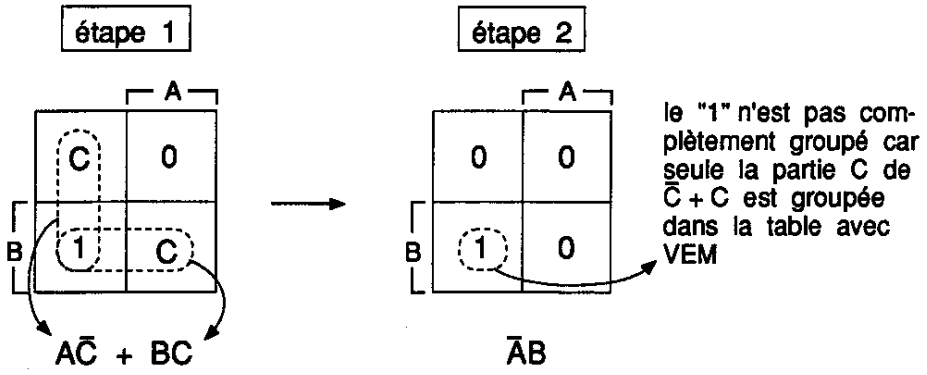
- c) *Remplacer les "1"*
  par des "1" - si la case n'est pas couverte complètement
  par des "X" - dans le cas contraire

  Remarque : Une case avec un "1" est couverte complètement lorsque les parties "$\bar{A}$" et "A" sont couvertes ($A + \bar{A} = 1$)

- d) *Remplacer les cases avec VEM composées avec un "X" par des "0"*
  CX, $\bar{C}$X, CDX ... $\longrightarrow$ *0*

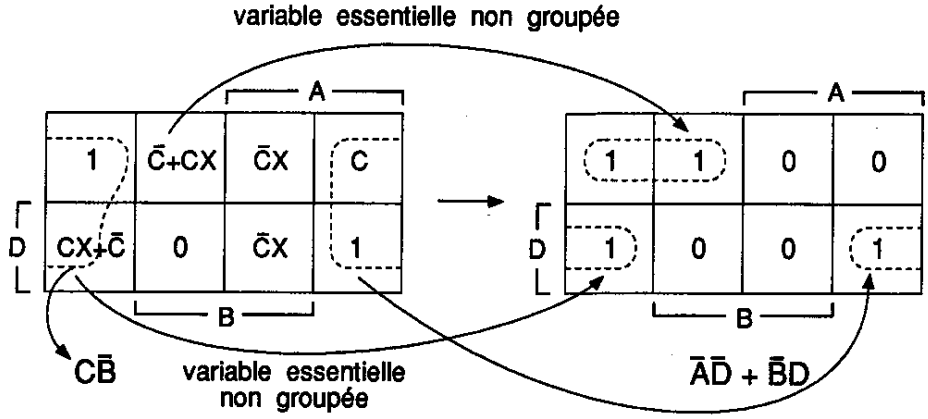
- e) *Remplacer les expressions de la forme*
  C + $\bar{C}$X ou $\bar{C}$ + CX
  par des "1" - si la variable essentielle est non groupée
  par des "X" - dans les autres cas
  dans
  C + $\bar{C}$X, "C" est la variable essentielle
  $\bar{C}$ + CX, "$\bar{C}$" est la variable essentielle

Exemple #1 :



Finalement : $F = \bar{A}C + BC + \bar{A}B$

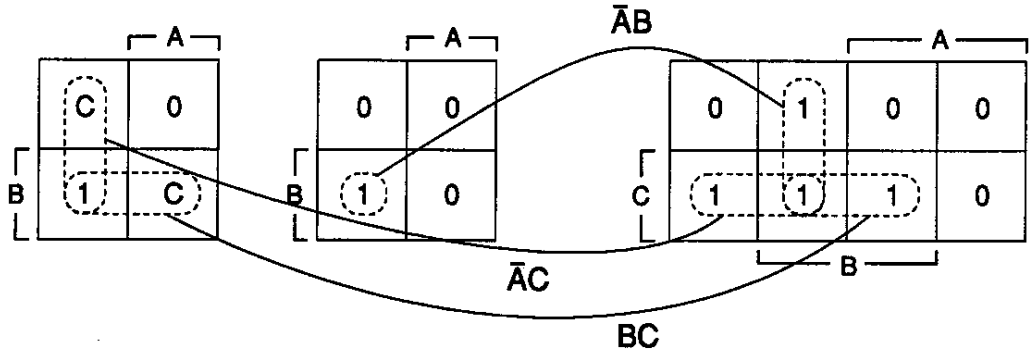
Exemple #2 :



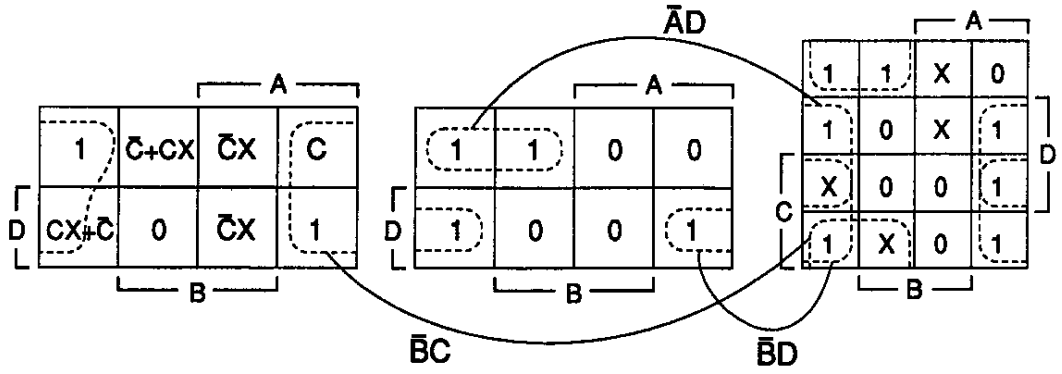
$$F = C\bar{B} + \bar{A}\bar{D} + \bar{B}D$$

Comparons les résultats des tables avec VEM aux tables complètes :

Vérification exemple #1 :

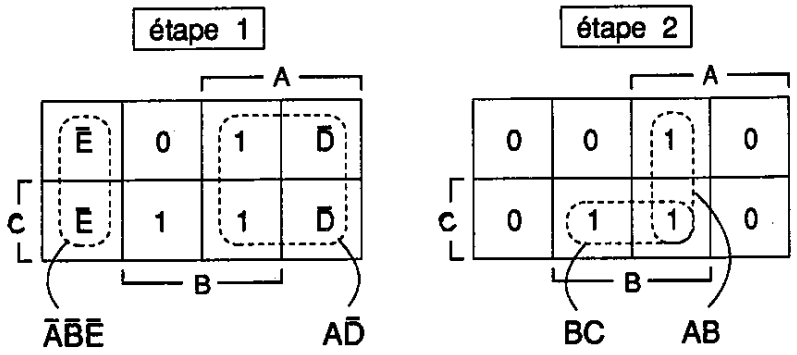


Vérification exemple #2 :



3.11.3 - Exemples avec plusieurs "VEM"

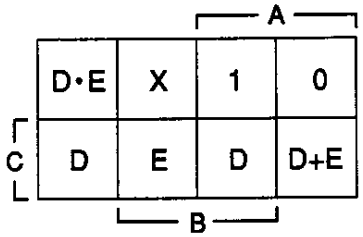
Solution d'une table à 5 variables avec 2 VEM



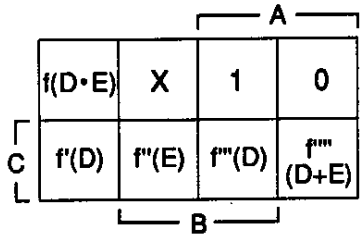
$$F = \bar{A}\bar{B}\bar{E} + A\bar{D} + BC + AB$$

3.11.4 - Les tables de Laurendeau (!)

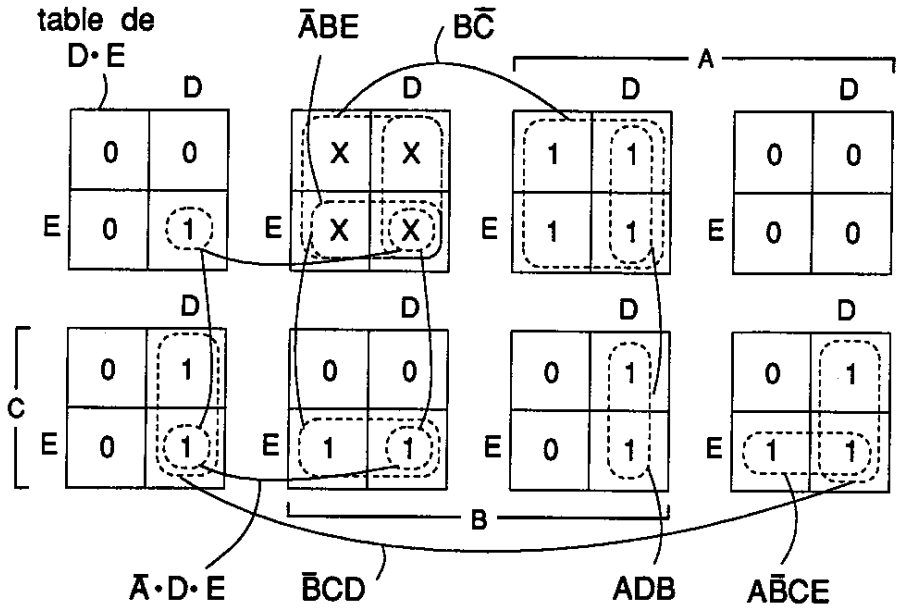
Jusqu'ici, l'application des règles de solution des tables de Karnaugh avec VEM s'est limitée à des cas simples où les VEM sont relativement bien disposées dans la table. Que faire lorsqu'on a une table vraiment complexe du genre.



La présence de $D \cdot E$ et $D + E$ dans les cases de la table rend très complexe l'utilisation des règles de solution de la table. Il faut cependant réaliser que chaque case avec VEM est une fonction de D et E.



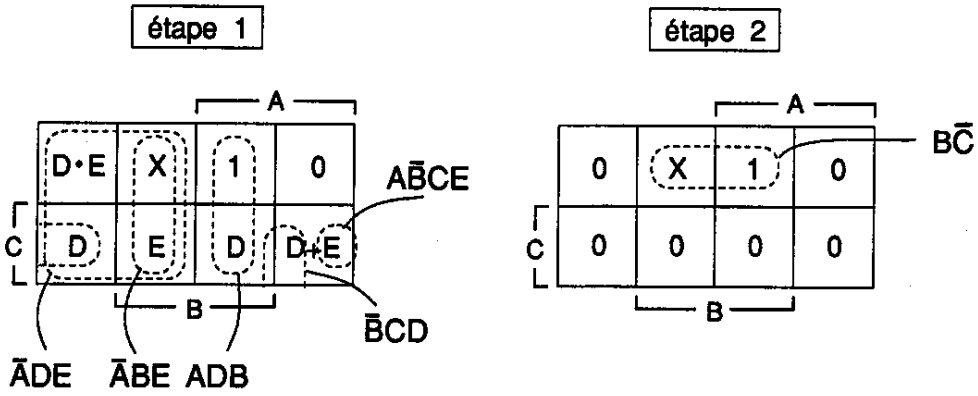
On peut adopter une représentation sous forme de table de Laurendeau pour solutionner la table. Cette représentation utilise des petites tables imbriquées dans des plus grosses. On solutionne les petites tables en premier, puis, on voit si l'on peut appliquer les termes (ou groupements) dans la grande table. Pour l'exemple à la page suivante, la table de Laurendeau s'écrit et se solutionne comme suit :



$$F = \bar{A}DE + \bar{B}CD + ADB + A\bar{B}CE + \bar{A}BE + B\bar{C}$$

Il est loin d'être certain qu'on serait arrivé à la même solution en solutionnant directement la table avec VEM en suivant les règles établies à la section 3.11.2.

On aurait dû faire :



$$F = \bar{A}DE + \bar{A}BE + ABD + A\bar{B}CE + \bar{B}CD + B\bar{C}$$

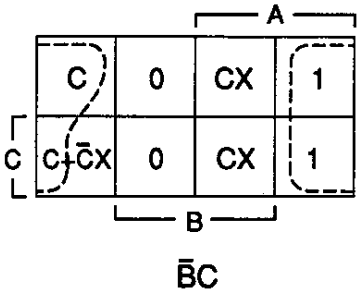
qui est bien ce qu'on a trouvé avec la table de Laurendeau.

Donc, lorsque la table est simple, on peut utiliser la technique avec VEM. Si la table est complexe, il vaut mieux prendre la table de Laurendeau qui n'est, en définitive, qu'une façon d'écrire une table de Karnaugh à plus de 6 variables sous une forme lisible. Avec la table de Laurendeau, on peut se rendre à 12 variables. Ce qui est suffisant pour la plupart des problèmes.

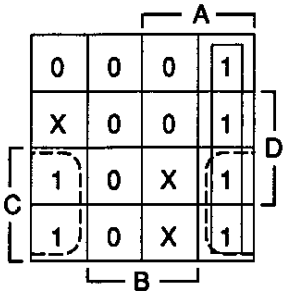
Explications supplémentaires concernant le remplacement des CX ou $\bar{C}X$ par des "0".

Soit le cas suivant :

étape 1

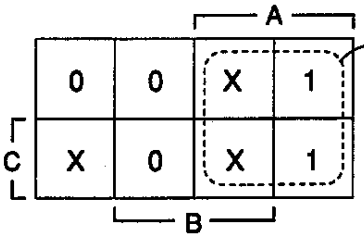


$\bar{B}C$

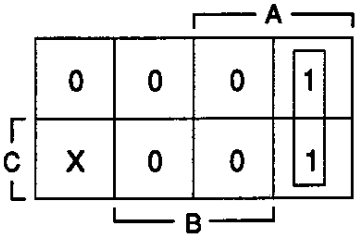


étape 2

à cette étape, si on remplacait les "CX" par des "X" on obtiendrait :



Ce qui nous permettrait de faire le groupement menant au terme "A", or, ce terme n'existe pas dans la table complète ci-dessus.



Par contre, avec les CX remplaçés par des "0", on arrive à la solution correcte :

$F = \bar{B}C + A\bar{B}$

3.11 - Variables conditionnelles (VEM) [autre version]

Les variables conditionnelles (VEM = "Variable Entered Mapping") produisent une extension de la table de Karnaugh. En effet, l'emploi des tables de karnaugh pour plus de 6 v.i. devient difficile pour ne pas dire impossible. Les "VEM" ajoutent à la puissance des tables de karnaugh en permettant plus de v.i.

Le principe des "VEM" est d'ajouter des v.i., les variables conditionnelles, qui attribuent une condition d'apparition du minterm dans l'expression de la fonction. Ainsi, la liste des conditions d'entrée dans une table de Karnaugh ne se limite plus qu'aux "1, 0 ou X", mais s'étend avec des conditions qui sont des variables ou même des expressions booléennes.

3.11.1 - Démonstration du procédé

Soit : $F = f(A,B,C)$ telle que :

A	B	C	F	minterm associé
0	0	0	f_0	$\bar{A}\bar{B}\bar{C}$ (m_0)
0	0	1	f_1	$\bar{A}\bar{B}C$ (m_1)
0	1	0	f_2	$\bar{A}B\bar{C}$ (m_2)
0	1	1	f_3	$\bar{A}BC$ (m_3)
1	0	0	f_4	$A\bar{B}\bar{C}$ (m_4)
1	0	1	f_5	$A\bar{B}C$ (m_5)
1	1	0	f_6	$AB\bar{C}$ (m_6)
1	1	1	f_7	ABC (m_7)

On remarque que, mathématiquement, chaque minterm est vectoriellement indépendant des autres. L'ensemble forme donc un espace.

$$F \cdot F = F$$

donc

$$m_0 f_0 + m_1 f_1 + \ldots m_7 f_7 = F$$

$$(\bar{A}\bar{B}\bar{C})\, f_0 + (\bar{A}\bar{B}C)\, f_1 + \ldots (\bar{A}\bar{B}C)\, f_7 = F$$

Vu sous cet angle, $f_0, f_1 \ldots f_7$ apparaissent comme des conditions d'apparition du minterm correspondant dans l'expression de "F".

Si $f_i = 0$ ⟶ le minterm n'apparait pas

Si $f_i = 1$ ⟶ le minterm est présent

Si $f_i = X$ ⟶ le résultat est indifférent

Ainsi, en prenant f_i comme une condition, rien n'empêche celui-ci d'être une variable indépendante autre que celles apparaissant dans les minterms ou encore une expression booléenne fonction de variables indépendantes des autres.

Par exemple, avec une mise en évidence, on obtient :

$$F = \bar{A}\bar{B}(\bar{C}f_0 + Cf_1) + \bar{A}B(\bar{C}f_2 + Cf_3) + A\bar{B}(\bar{C}f_4 + Cf_5) + AB(\bar{C}f_6 + Cf_7)$$

La fonction F peut ainsi se simplifier sur une table de Karnaugh à 2 variables, A et B, ou chaque terme entre parenthèse est la condition d'apparition du sub-minterm correspondant. Cependant, la variable C apparaît maintenant aussi dans la condition d'apparition du minterm dans la table de Karnaugh à 2 variables.

Exemple #1 :

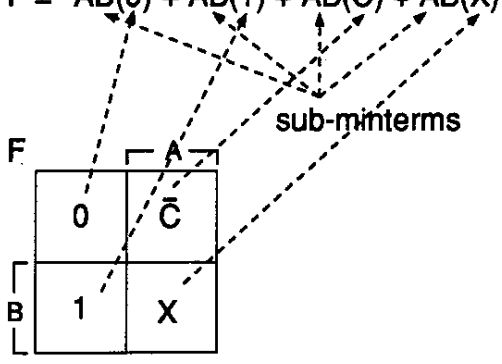
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	X
1	1	1	X

$$F = f(A,B,C) = \bar{A}\bar{B}\bar{C}f_0 + \bar{A}\bar{B}Cf_1 + \bar{A}B\bar{C}f_2 + \bar{A}BCf_3 + A\bar{B}\bar{C}f_4 + A\bar{B}Cf_5 + AB\bar{C}f_6 + ABCf_7$$

$$= \bar{A}\bar{B}(\bar{C}f_0 + Cf_1) + \bar{A}B(\bar{C}f_2 + Cf_3) + A\bar{B}(\bar{C}f_4 + Cf_5) + AB(\bar{C}f_6 + Cf_7)$$

$$F = \bar{A}\bar{B}(0) + \bar{A}B(\bar{C}+C) + A\bar{B}(\bar{C}) + AB(\bar{C}X + CX)$$

$$F = \bar{A}\bar{B}(0) + \bar{A}B(1) + A\bar{B}(\bar{C}) + AB(X)$$



Exemple #2 :

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	0
0	1	1	0	X
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	X
1	1	0	1	X
1	1	1	0	0
1	1	1	1	0

F : fonction de 4 variables ind. A,B,C,D

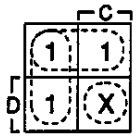
C = variables conditionnelle (choix arbitriare)

A,B,C = variables indépendantes du sub-minterm

$F = \bar{A}\bar{B}\bar{D}(1) + \bar{A}\bar{B}D(\bar{C}+CX) + A\bar{B}\bar{D}(\bar{C}+CX) + \bar{A}BD(0) + A\bar{B}\bar{D}(C) + A\bar{B}D(1) + AB\bar{D}(\bar{C}X) + ABD(\bar{C}X)$

F	A			
	1	$\bar{C}+CX$	$\bar{C}X$	C
D	$\bar{C}+CX$	0	$\bar{C}X$	1
L		B		

On peut prendre 2 variables conditionnelles (C et D) laissant 2 variables indépendantes pour composer les sub-minterms (A et B).



F	A	
	$\bar{D}+\bar{C}+CDX$	C + D
B	$\bar{C}\bar{D}+C\bar{D}X$	$\bar{C}X$
L		

Avec la symbologie adaptée de la façon suivante :

$CX \begin{cases} = 0 \text{ lorsque } C = 0 \\ = X \text{ lorsque } C = 1 \end{cases}$

$\bar{C}X \begin{cases} = X \text{ lorsque } C = 0 \\ = 0 \text{ lorsque } C = 1 \end{cases}$

$\bar{C}+CX \begin{cases} = X \text{ lorsque } C = 0 \\ = 0 \text{ lorsque } C = 1 \end{cases}$

etc ...

3.11.2 - Lecture avec VEM

Une fois la table de Karnaugh complétée, encore faut-il la lire pour la simplification. Ici, cependant, la lecture avec des variables conditionnelles n'est pas une tâche facile. Il y a plusieurs étapes à suivre.

Étape 1 : La couverture de toutes les variables conditionnelles dans la table. Les termes produits dans cette étape contiennent tous au moins une conditionnelle (VEM).

- a) *Encercler toutes les VEM simples qui ne peuvent être groupées avec une autre VEM identique adjacente,avec un "1" ou un "X"*

  Remarque : - C et $\bar{C}$ ne sont pas identiques
  - CX n'est pas un VEM simple

- b) *Faire les groupements de cases avec VEM identiques et adjacentes.*

- c) *Faire les groupements de 2 cases avec VEM dans l'une et un "1" dans l'autre.*

- d) *Faire les groupements de 2 cases avec VEM dans l'une et un "X" dans l'autre.*

  Remarque : Toutes les VEM pouvant se grouper de plusieurs façons avec une VEM identique et adjacente, un "1" ou un "X", mais sans se grouper dans un groupement plus gros (4, 8, ...) doivent être groupées en dernier lieu avec une attention toute spéciale.

- e) *Répéter ces sous-étapes pour effectuer les groupements de 4,8 ... cases.*

Étape 2 : Lorsque toutes les VEM simples (sans qu'elles soient multipliées par X) sont regroupées, il faut transformer la table pour les termes de l'expression qui ne contiendront pas une variable conditionnelle. les remplacements sont les suivants :

- a) *Remplacer les VEM simples par des "0"*
- b) *Remplacer les* → *"0" par des "0"*
  → *"X" par des "X"*
- c) *Remplacer les "1"*
  par des "1" si non couvert complètement
  par des "X" dans le cas contraire.

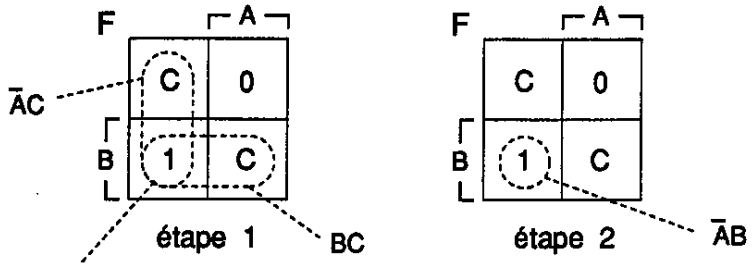
  Remarque : on sait que $1 = A+\bar{A}$, si la case marquée d'un "1" a servi pour des groupements avec A et $\bar{A}$ alors le "1" est complètement couvert.

- c) *Remplacer les variables conditionnelles composées avec un "X" par des "0"*
  CX, $\bar{C}$X, CDX ...

- e) *Remplacer les expressions de la forme*
  (C + $\bar{C}$X) ou ($\bar{C}$ + CX)
  par des "1" si la variable essentielle est non groupée
  par des "X" dans le cas contraire
  v. essentielle

Étape 3 : Terminer la lecture de la nouvelle table comme d'habitude.

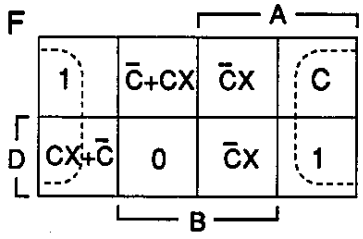
Exemple #1 :



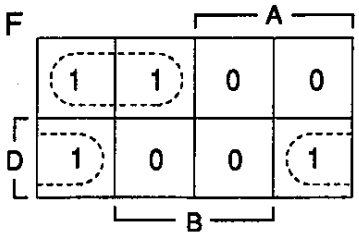
pas complètement couvert car si on décompose $1 = C + \bar{C}$ seul C a servi aux regroupements

$F = \bar{A}C + BC + \bar{A}B$

Exemple #2 : on fait la lecture de l'exemple #2 du paragraphe 3.11.1



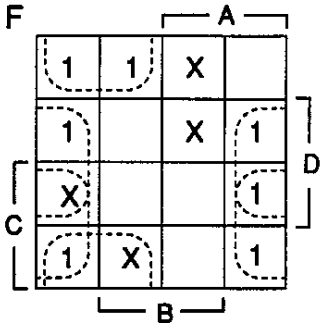
étape 1



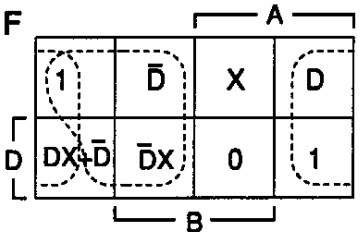
étape 2

$F = \bar{B}C + \bar{A}\bar{D} + \bar{B}D$

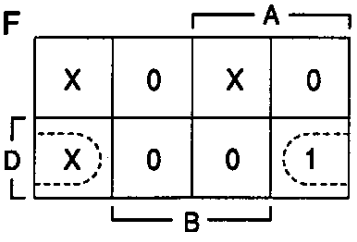
vérification



Exemple #3 : si, au lieu de prendre "C" comme variable conditionnelle pour l'exemple #2 du paragraphe 3.11.1, on avait pris "D", on obtiendrait ...



étape 1



étape 2

$F = \bar{B}D + \bar{A}\bar{D} + \bar{B}C$

... la même réponse

3.11.3 - Avantages des VEM

La méthode des "VEM" peut sembler fort difficile d'usage mais elle gagne de l'efficacité dans des conditions qui, en réalité, apparaissent dans un grand nombre de cas. En effet, les "VEM" sont d'autant plus avantageuses que lorsque des variables apparaissent peu fréquemment ou mieux des variables sont actives (haut ou bas) que très rarement.

Dans ces 2 possibilités, les variables rares sont traitées comme des variables conditionnelles et n'ont pas à être placées "autour" de la table de Karnaugh.

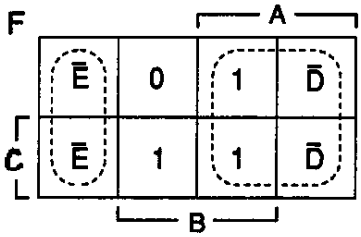
Exemple : variables peu fréquentes

$F = f(A,B,C,D,E)$

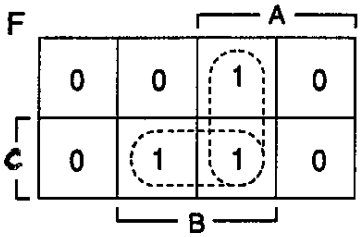
$F = \bar{A}\bar{B}\bar{C}\bar{E} + \bar{A}\bar{B}C\bar{E} + \bar{A}BC + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C} + ABC$

Les variables D et E sont infréquemment utilisées, on peut donc les placer comme variables conditionnelles.

$F = \bar{A}\bar{B}\bar{C}(\bar{E}) + \bar{A}\bar{B}C(\bar{E}) + \bar{A}B\bar{C}(0) + \bar{A}BC(1) +$
$A\bar{B}\bar{C}(\bar{D}) + A\bar{B}C(\bar{D}) + AB\bar{C}(1) + ABC(1)$



étape 1



étape 2

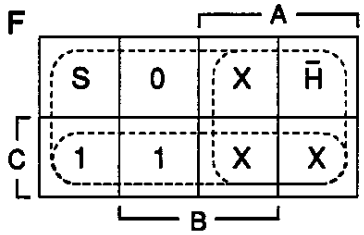
$F = \bar{A}\bar{B}\bar{E} + A\bar{D} + AB + BC$

Exemple : variables actives rarement

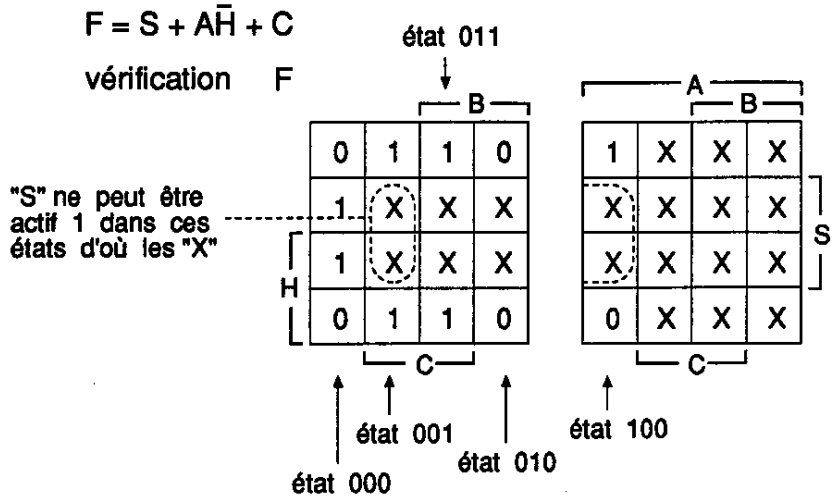
Dans les systèmes dits séquentiels, il arrive souvent que des commandes soient actives dans certains états de la machine. Les états se définissent par un code binaire où les bits du code sont des variables indépendantes. Ainsi, lorsque les commandes, qui sont aussi des v.i., sont actives rarement, il est préférable de traiter les commandes comme des variables conditionnelles et les bits du code comme les variables des sub-minterms.

Un système à 5 états codés sur 3 bits (A,B,C) possède 2 commandes extérieures, "start" (S) et halte (H). Or, on suppose que le signal "start" ne peut être actif que dans l'état de départ codé 000 (actif 1). Quant au signal "halte", il peut se produire n'importe quand mais il n'agit que dans le dernier état codé 100.

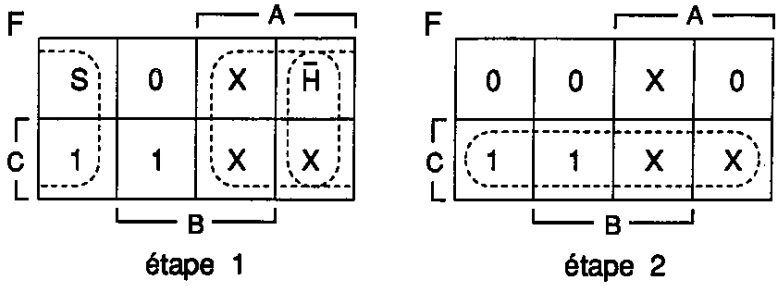
On obtient la table de Karnaugh suivante lors de l'obtention d'une fonction interne de la machine :



Puisque "S" ne peut valoir "1" dans un autre état que l'état 000 alors "S" vaut "0" dans les états 010 et 100. Par contre H peut être égal à "0" ou à "1" dans tous les états. On procède donc pour H de la même façon que vue précédemment. L'histoire est différente pour "S" avec lequel on peut grouper la table au complet n'ayant pas d'influence dans les autres états.

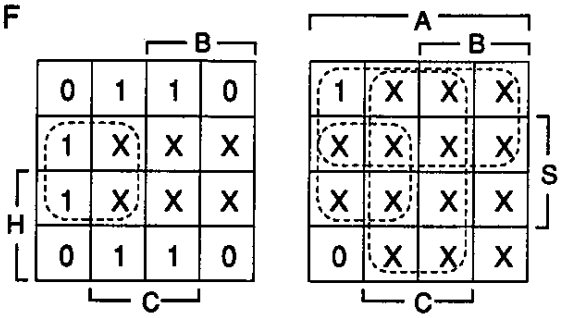


Par contre, avec "S" actif haut possiblement dans les états 000, 001, et 010, on obtient :



$F = \bar{B}S + A\bar{H} + C$

vérification F



Le tout sera discuté plus fermement plus tard.

3.12 - synthèse avec "MSI"

Les circuits "MSI" (Medium Scale Integration) les plus répandus sont :

- les multiplexeurs (74150, 74151, 74153, 74157 ...)
- les décodeurs (7442, 74138, 74154, 74155 ...)
- les encodeurs de priorité (74148)
- les générateurs de parité (74180)
- les additionneurs (7483), multiplicateurs (74284)
- les comparateurs (7485)
- les compteurs, les régistres, les bascules ...

L'avantage de l'utilisation des circuits "MSI" repose sur le fait que certaines fonctions logiques existent dans un seul boitier. Ils réduisent la quantité de câblage à faire et l'espace occupé et ce en conservant un coût identique quand ce n'est pas le moindre.

La synthèse classique des circuits combinatoires, telle que vue précédamment, est une méthode à utiliser en dernier recours pour des fonctions relativement complexes. Il faut donc, dès le départ, penser au choix à faire : "MSI" ou "SSI"? Rien de mieux qu'une bonne analyse avant de prendre la décision et une bonne connaissance du catalogue ...

3.12.1 - synthèse par multiplexeur

Définition : Circuit combinatoire d'aiguillage à 2^n entrées d'information, n entrées adresse et une seule sortie. La sortie correspond à l'entrée d'information de rang égal à la valeur binaire de l'adresse présente.

Exemple : le multiplexeur (MUX) 8 à 1, le 74151

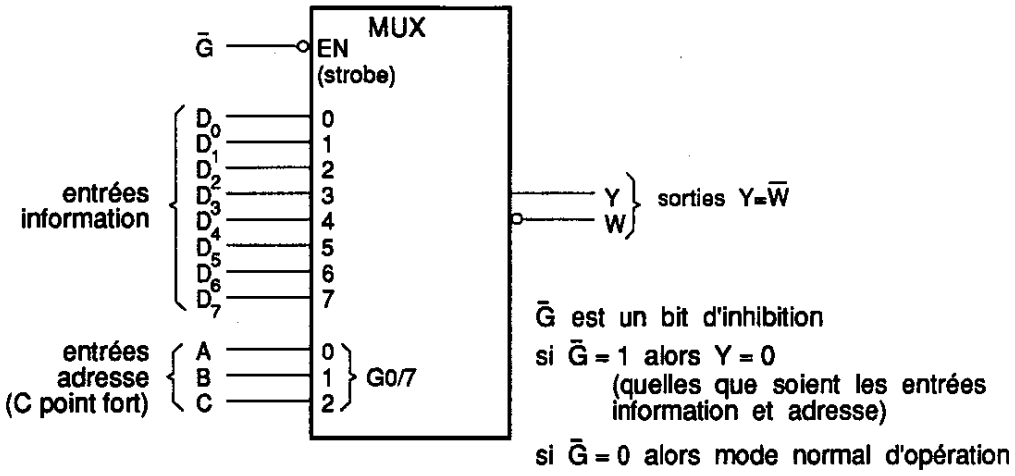


table de vérité

C	B	A	$\bar{G}$	$Y(W=\bar{Y})$	
X	X	X	1	0	← quels que soient C, B et A avec $\bar{G} = 1$, $Y = 0$
0	0	0	0	D_0	
0	0	1	0	D_1	
0	1	0	0	D_2	
0	1	1	0	D_3	
1	0	0	0	D_4	
1	0	1	0	D_5	
1	1	0	0	D_6	
1	1	1	0	D_7	

Réalisation de principe :

L'expression générale de la sortie d'un multiplexeur est :

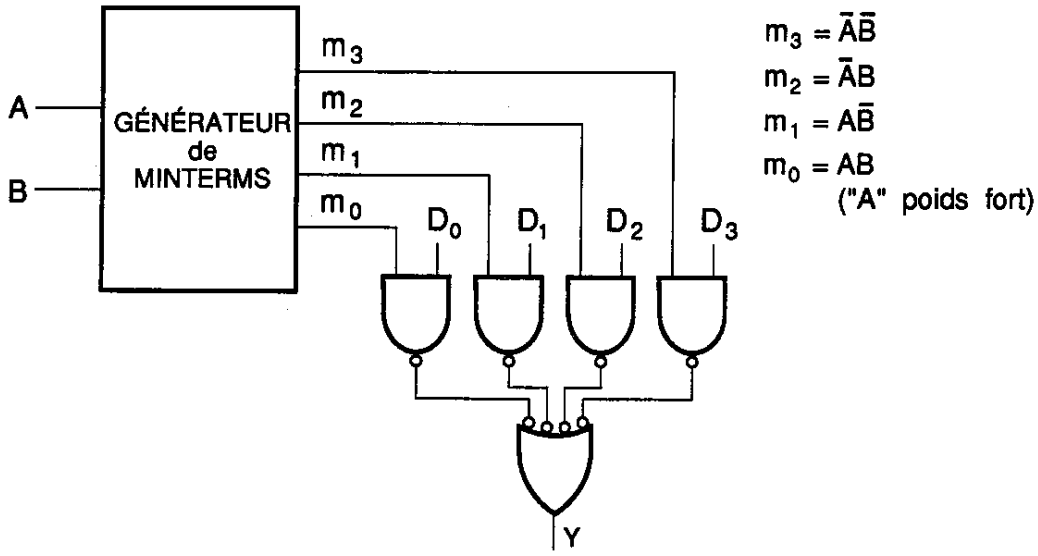
$Y = m_0D_0 + m_1D_1 + \ldots m_{2^n-1}D_{2^n-1}$

ou

m_j est un minterm construit avec les entrées adresse
D_j est un entrée information de rang j

Il faut donc produire tous les minterms possible avec les entrées adresse; multiplier les minterms avec les entrées information correspondantes; sommer le tout.

SCHÉMA FONCTIONNEL



La famille TTL compte plusieurs multiplexeurs de toutes les capacités. (La capacité ou dimension d'un multiplexeur représente le nombre d'entrées information pour chaque sortie) (i.e. le nombre de bits d'adressage des sorties).

75150 ⟶ MUX 16 à 1 (boitier à 24 broches : coût $1^1/_2$ boitiers)
75151 ⟶ MUX 8 à 1
75153 ⟶ 2 MUX 4 à 1
75157 ⟶ 4 MUX 2 à 1

Les principales applications du multiplexeur sont :

- *l'aiguillage ie la sélection d'une bit*
- *la conversion parallèle-série*
- *la génération de fonctions combinatoires*

Génération de fonctions combinatoires :

C'est le cas qui nous intéresse le plus pour le moment. L'analyse du multiplexeur et de l'expression de la sortie montre que celui-ci agit de manière à attribuer une condition d'apparition à chaque minterm un peu comme le fonctionnement des VEM. On n'a pas à se surprendre d'une certaine liaison entre les "MUX" et les "VEM".

soit un "MUX" 4 à 1 avec les entrées adresse A et B
soit F = f(A,B,C,D,E)

on décompose l'expression de F de façon à obtenir quelque chose de la forme

$F = \bar{A}\bar{B}[R_0(C,D,E)] + \bar{A}B[R_1(C,D,E)] + A\bar{B}[R_2(C,D,E)] + AB[R_3(C,D,E)]$

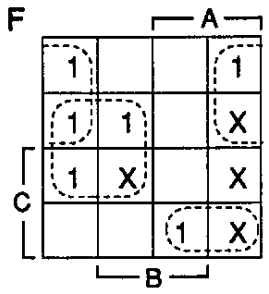
où

$R_i(C,D,E)$ = fonction des variables résiduelles C,D,E qui ressemblent à des variables conditionnelles.

Ainsi, avec l'utilisation d'un MUX 4 à 1, on réduit de 2 le nombre de variables avec lesquelles on doit trouver une expression logique simplifiée. On est rendu à faire 4 synthèses d'expression avec 2 variables indépendantes en moins. Plus généralement, un MUX 2^n à 1 simplifie la synthèse de n variables bien qu'il faut alors résoudre 2^n fonctions au lieu d'une. Le jeu se résume à bien sélectionner la dimension du multiplexeur.

Exemple : en utilisant la technique des VEM

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	X
1	0	1	0	X
1	0	1	1	X
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



$F = AC\bar{D} + \bar{A}D + \bar{B}\bar{C}$ coût : $\frac{1}{3} + \frac{1}{3} + \frac{1}{4} + \frac{1}{4} : 1\frac{1}{16}$

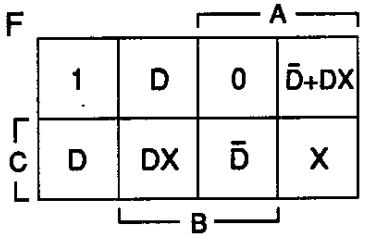
(requiert 2 circuits intégrés un 7410, un 7400)

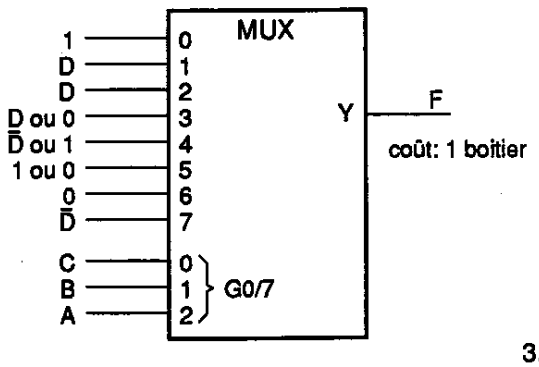
avec un MUX 16 à 1 coût = $1\frac{1}{2}$ mais <u>aucun</u> câblage et un seul circuit intégré 74150!

avec un MUX 16 à 1, les 4 variables A,B,C et D servent comme variables adresses. Dans ce Cas, il ne reste aucune variable résiduelle avec laquelle on construit les 16 fonctions résidus (Ri). Les entrées information sont donc des "1" ou des "0". Par exemple : D14 = 1, D6 = 0, D5 = 1, D4 = 0 etc...

avec un MUX 8 à 1, 3 variables adresse. Les 8 fonctions "résidus" ne sont donc dépendantes que d'une seule variable.

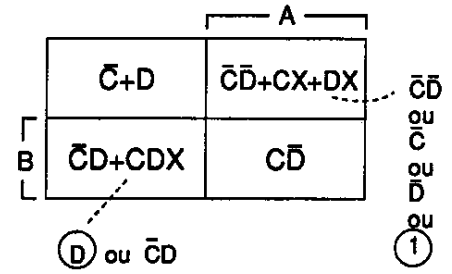
A, B et C = variables adresse
D = variable résiduelle (ou conditionnelle)

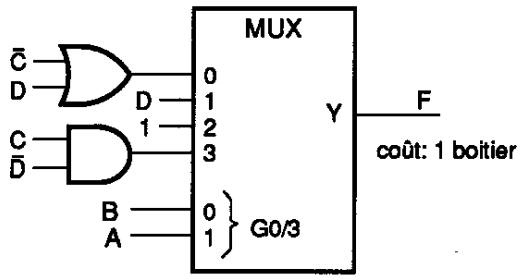




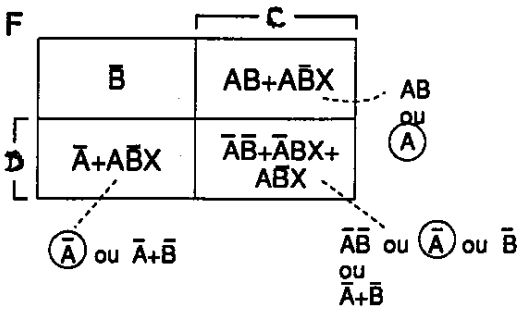
avec un MUX 4 à 1, 2 variables adresse. Les 4 fonctions résidus dépendent des 2 autres variables Il importe de connaître quelles sont les 2 variables adresse qui simplifient le plus les fonctions résidus.

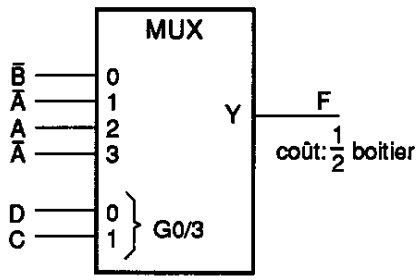
A,B = variables adresse
C,D = variables résiduelles





C,D = variables adresse
A,B = variables résiduelles





En procédant de la même façon que les "VEM", la conception de circuits combinatoires avec multiplexeurs semble imprévisible : comment choisir la dimension des "MUX" et comment sélectionner les bonnes variables adresse? La réponse aux deux questions ne suit pas une règle infaillible, il faut y aller avec plusieurs essais mais, bientôt, l'expérience vous guidera. On peut s'accommoder d'outils comme la division d'une table de Karnaugh en sous-tables dépendantes des variables résiduelles seulement. Cette technique des sous-tables peut s'appliquer aussi aux "VEM".

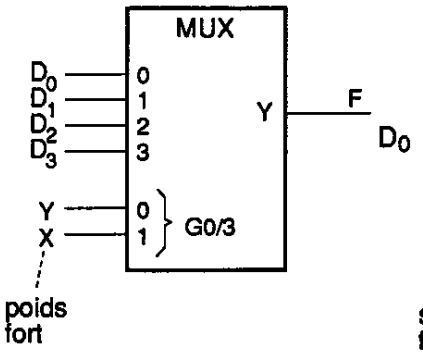
Choix des var. adresse :
(technique des sous-tables)

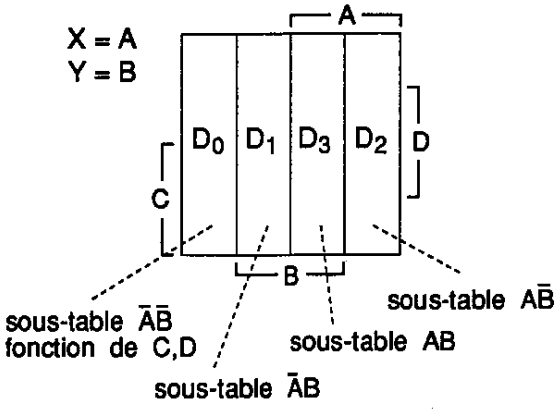
* *Diviser la table de Karnaugh en 2^n sous-tables avec n = nombre d'entrées adresse.*

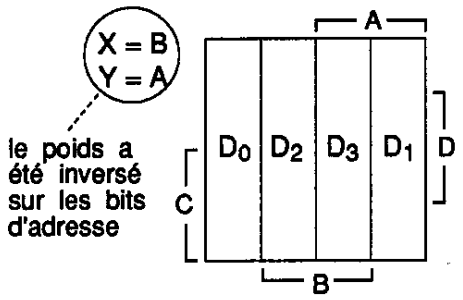
  Chaque sous-table est associée à une entrée information. L'arrangement de chaque sous-table est tel que seules les limites des zones des variables résiduelles traversent la sous-table

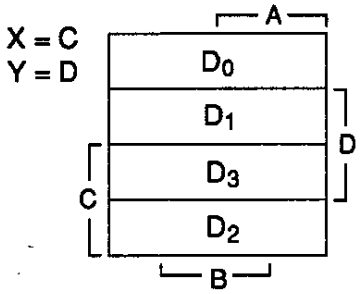
* *Résoudre chaque sous-table fonction des variables résiduelles seulement.*

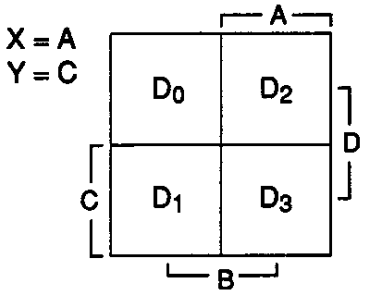
Exemple : arrangement des sous-tables dans une table de Karnaugh à 4 v.i. en utilisant un MUX 4 à 1.

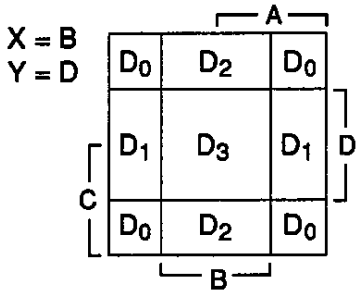










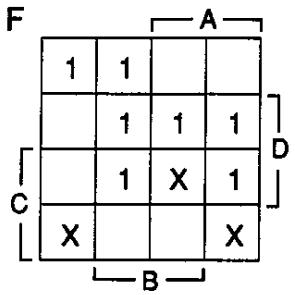


Note : - faire attention au poids des var. adresse
- ici, chaque sous-table est fonction de 2 v.i.

Exemple #1 : F = f(A,B,C,D)

F = Σ (0,4,5,7,9,11,13) X (2,10,15) , MUX 4 à 1

Exemple #1 (suite)...



avec AB comme adresse
$D_0 = \bar{D}$ $D_2 = D$
$D_1 = \bar{C}+D$ $D_3 = D$

avec AC comme adresse
$D_0 = B+\bar{D}$ $D_2 = D$
$D_1 = BD$ $D_3 = D$

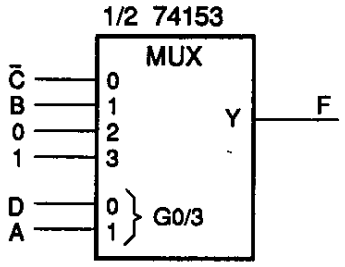
avec BD comme adresse
$D_0 = \bar{A}$ $D_2 = \bar{A}\bar{C}$
$D_1 = A$ $D_3 = 1$

avec CD comme adresse
$D_0 = \bar{A}$ $D_2 = 0$
$D_1 = A+B$ $D_3 = A+B$

avec BC comme adresse
$D_0 = \bar{A}\bar{D}+AD$ $D_2 = \bar{A}+D$
$D_1 = A$ $D_3 = D$

avec AD comme adresse
$D_0 = \bar{C}$ $D_2 = 0$
$D_1 = B$ $D_3 = 1$

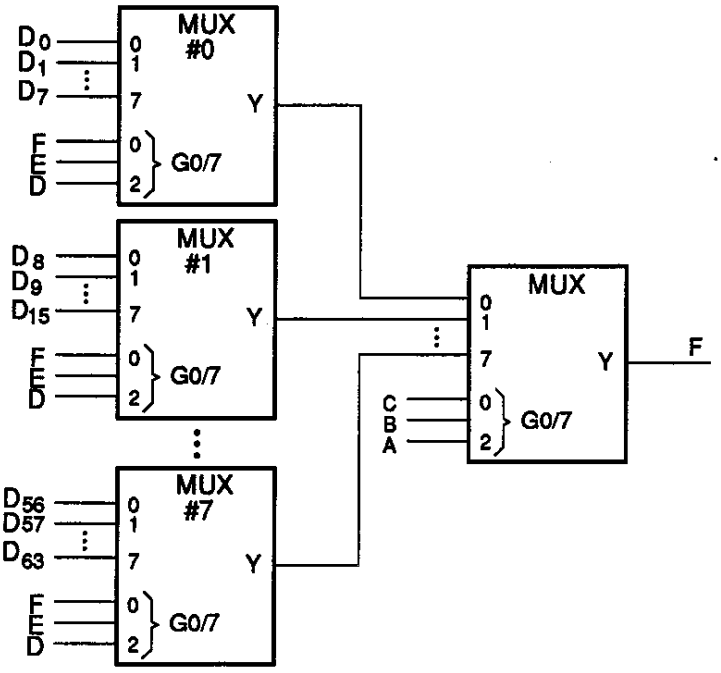
La meilleure possibilité est obtenue avec A et D comme adresse. Le coût se résume alors à celui du multiplexeur 4 à 1 (1/2 boitier).



Expansion du MUX :

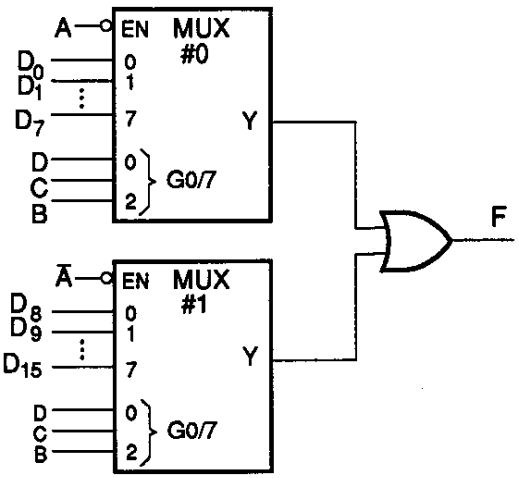
- Par couches (arbre) MUX 64 à 1

  Le MUX de la 2e couche sélectionne l'une des sorties des MUX de la 1ère couche par le biais des bits d'adresse de poids fort



- Avec bit d'inhibition
  MUX 16 à 1

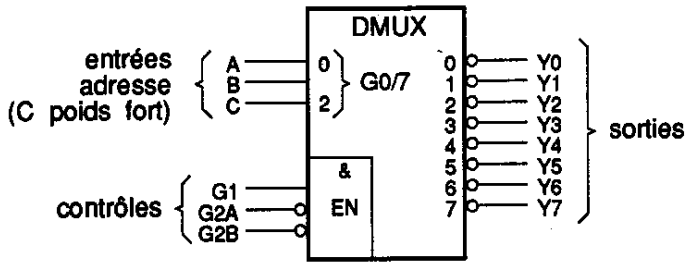
  La bit d'adresse de poids fort (A) sélectionne le MUX en mode normal



3.12.2 - Synthèse par décodeur

Définition : *Circuit combinatoire à n entrées adresse et 2^n sorties où une seule est activée à la fois. Le rang de la sortie activée correspond à la valeur binaire de l'adresse présente.*

Exemple : le décodeur (X/Y ou DMUX) 3 à 8, le 74138



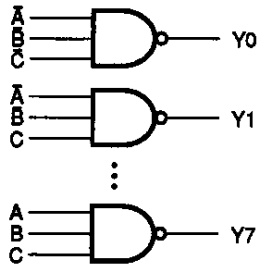
Les 3 bits d'inhibition agissent ensemble si $(G1 \cdot \overline{G2A} \cdot \overline{G2B}) = 1$ alors mode normal d'opération sinon toutes les sorties $Y_i = 1$.

table de vérité

G1	(G2A+G2B)	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	1	X	X	X	1	1	1	1	1	1	1	1
0	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

Réalisation de principe :

L'examen du décodeur montre que ce dernier active une sortie à la fois, sortie dépendant de l'adresse, tout comme les minterms, des combinaisons possibles des variables indépendantes. Le décodeur génère donc les minterms (certains décodeurs, ne fonctionnant pas sur le principe de la définition, produisent un code à la sortie dépendant du code présenté à l'entrée, d'où l'abréviation X/Y. Le 7442 est un décodeur BCD à décimal).



A, poids fort

$Y0 = \bar{m}_0$

$Y1 = \bar{m}_1$

$Y7 = \bar{m}_7$

La famille TTL compte plusieurs décodeurs (DMUX et X/Y) de quelques capacités ou dimensions. La dimension d'un décodeur fait appel au nombre d'entrées adresse et au nombre de sorties correspondantes.

7442 → X/Y BCD à décimal
7443 → X/Y Excess 3 à décimal
74154 → DEMUX 4 à 16 (boitier à 24 broches : coût $1\frac{1}{2}$ boitier)
74138 → DEMUX 3 à 8
74155 → 2 DEMUX 2 à 4

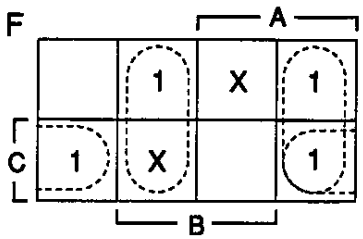
Note : Ils ont tous une ou des bits d'inhibition qui peuvent servir d'entrée pour démultiplexer et faire l'expansion.

Génération de fonctions combinatoires :

Le principe de base de l'expression d'une fonction combinatoire consiste à sommer les monômes canoniques, les minterms, associés aux combinaisons des v.i. auxquelles la fonction est active. Comme le décodeur produit tous les minterms, il est facile de comprendre la génération de fonctions combinatoires avec décodeurs. Il suffit donc de prendre les minterms nécessaires et produits par le décodeur; de les sommer en les inversant préalablement comme pour une expression du type SOP.

Exemple #1 : $F = f(A,B,C)$

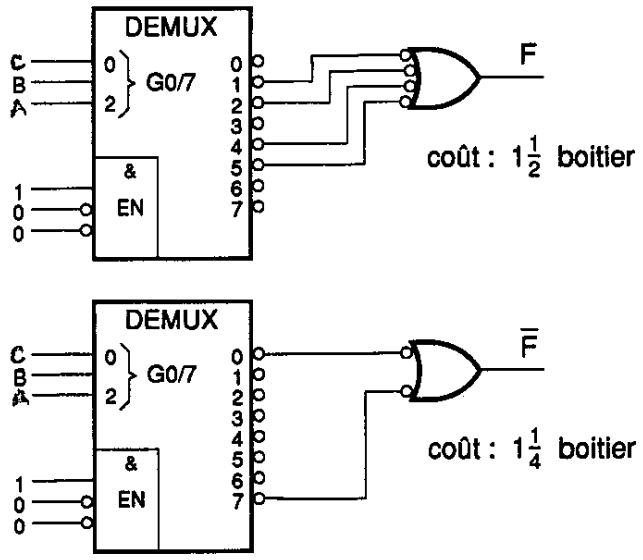
$F = \Sigma\, m(1,2,4,5)\ X\ (3,6)$



$F = A\bar{B} + \bar{A}B + \bar{B}C$

synthèse classique coût : $1\frac{1}{12}$

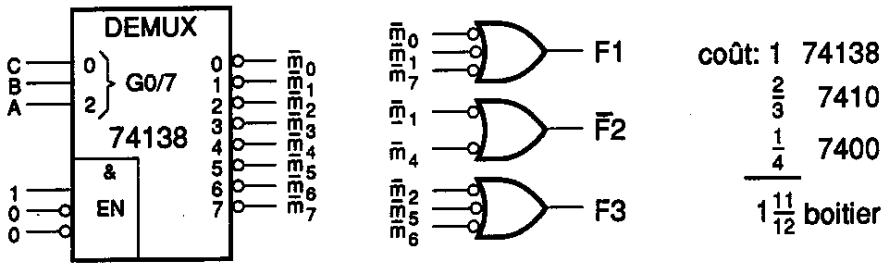
Avec un décodeur 3 à 8



Il vaut mieux faire la synthèse par les "0" car il y a moins de "0" que de "1" dans la table de vérité.

L'exemple précédent à tendance à montrer qu'il semble plus coûteux d'utiliser un décodeur que la synthèse avec portes. Pour une fonction simple, c'est souvent le cas mais pour des multifonctions (ou fonctions simultanées), le décodeur montre sa supériorité. Il va sans dire qu'il faut toutefois mêler les synthèses avec "SSI" et "MSI" dépendamment des fonctions à réaliser. En exercice, vous pouvez toujours essayer la synthèse du décodeur BCD-7 segments avec multiplexeurs et décodeurs (coût : $3\frac{7}{12}$ boitiers).

Exemple #2 : F1 (A,B,C) = Σm (0,1,7) X (2,4)
F2 (A,B,C) = Σm (0,2,3,5) X (6,7) } MULTIFONCTIONS
F3 (A,B,C) = Σm (2,5,6) X (3)

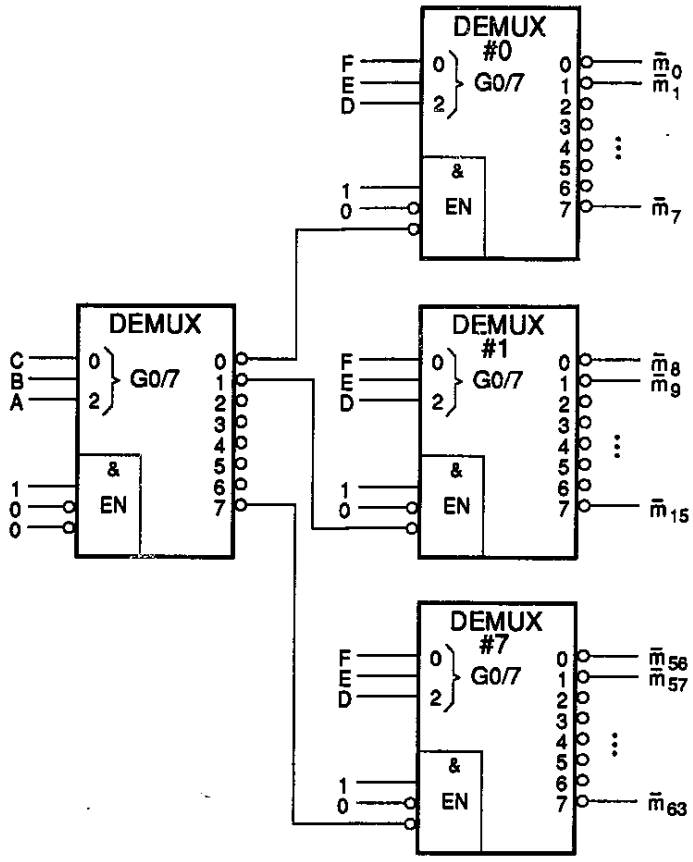


Avec des portes, il est difficile de faire mieux (impossible même) et ce, sans tenir compte du peu de câblage qu'exige le décodeur.

Expansion du DEMUX :

- Par couches (arbre)
  DEMUX 6 à 64 (6 à 2^6)

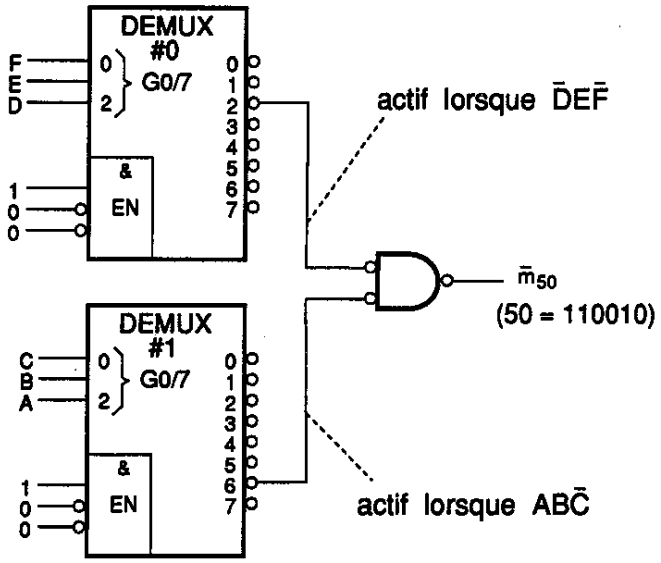
  Le DEMUX de la 1ère couche sélectionne un des DEMUX de la 2e couche par les bits d'adresse de poids fort. La sortie activée du DEMUX sélectionnée dépend alors des bits d'adresse de poids faible.



- Par matrice DEMUX 6 à 64

  exemple de formation d'un minterm en particulier

  Le DEMUX avec les bits d'adresse de poids fort forme les MSB du minterm et l'autre DEMUX, forme les LSB du minterm. Il s'agit d'activer le minterm lorsque les 2 "sous-minterms" sont actifs simultanément.



3.13 - Encodeur de priorité ("Priority Encoder")

Définition : *Circuit combinatoire d'encodage à 2^n entrées information et n sorties adresse. Les sorties donnent la valeur binaire du rang de l'entrée active la plus prioritaire parmi toutes les entrées actives. Généralement, on rajoute une sortie complémentaire ("Enable Output : EO") qui indique qu'au moins une entrée est active.*

Exemple : L'encodeur 8 à 3, le 74148

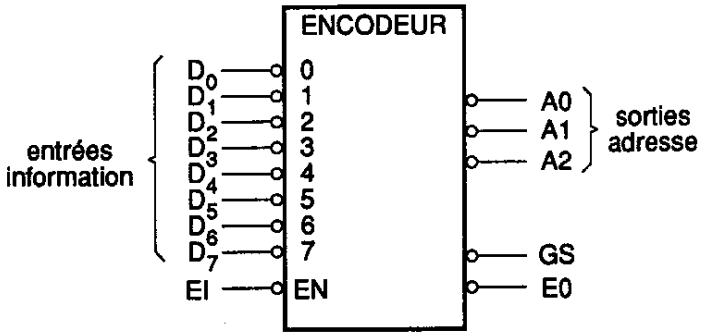


table de vérité

EI	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A2	A1	A0	GS	EO
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

Les encodeurs de priorité ne servent pas souvent à la genèse de circuits combinatoires car ils ne peuvent pas à proprement parler générer les monômes canoniques ("minterms")

3.14 - Aléas

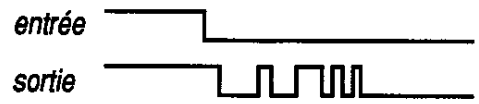
3.14.1 - Définition

Les aléas sont des malfonctionnements de circuits combinatoires et leurs commandes qui causent des erreurs transitoires ("glitches"). Ils surviennent lorsqu'une entrée change.

Les aléas proviennent de plusieurs sources et, dépendamment de la source et de leur effet, on les classe en 3 groupes principaux :

- aléas statiques : *aléas causés par les délais inégaux de 2 ou plus voies de transmission à travers un circuit combinatoire. Ils provoquent une transition simple et momentanée du signal de sortie.*

- aléas dynamiques : *Transitions multiples mais momentanées du signal de sortie en réponse à un changement d'une entrée. Les plus connus et répandus sont ceux provenant d'un contact mécanique qui rebondit (cf amortissement avec cellule binaire dans le prochain chapître).*

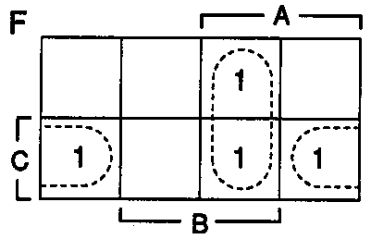


- aléas essentiels : *erreurs d'opération qui provoquent des transitions dans des états non valides en réponse à un changement d'état ou de commande dans un système séquentiel asynchrone. (chapître 9 - non au programme du cours).*

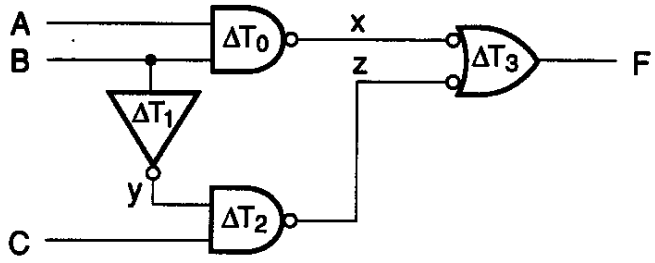
3.14.2 - Aléas statiques

Un aléa statique apparait dans une table de Karnaugh comme un saut d'un groupement à un autre lorsqu'on change une et une seule des v.i. du circuit. La transition simple qu'il crée ne dure qu'un infime laps de temps correspondant, en fait, à la différence de propagation des voies de transmission de l'entrée à la sortie.

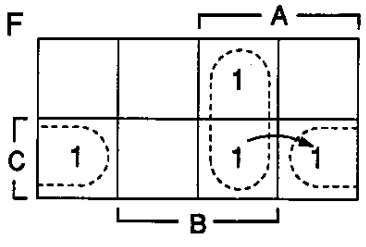
Exemple : $F = \Sigma m\,(1,5,6,7)$



$F = AB + \bar{B}C$

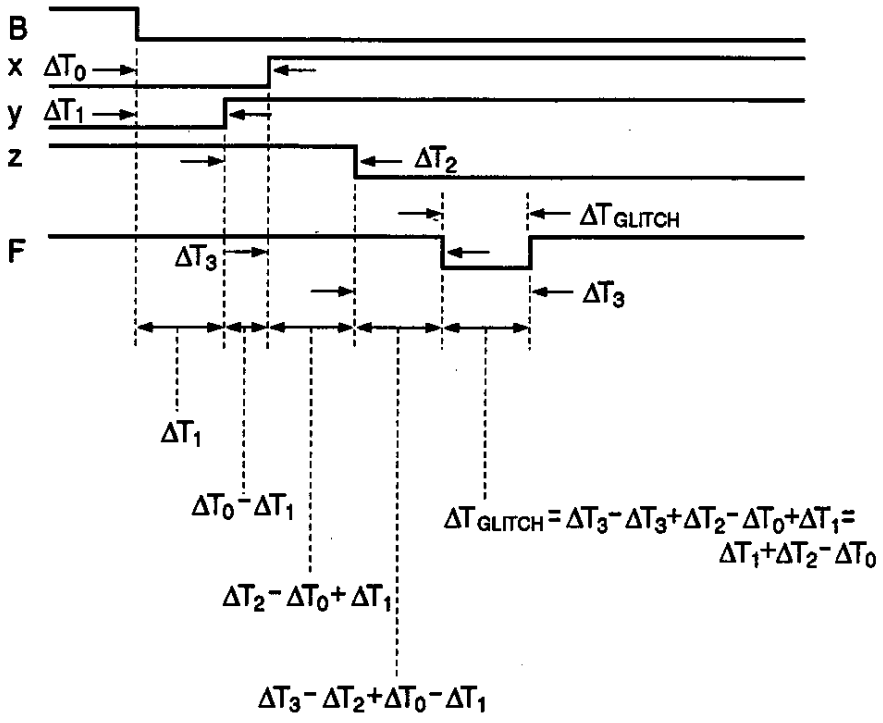


Regardons ce qui se passe lorsque A = C = 1 et que B fait la transition de "1" à "0"



$F = \Sigma m(1,5,6,7)$

$F = AB + \bar{B}C = \overline{x \cdot z}$



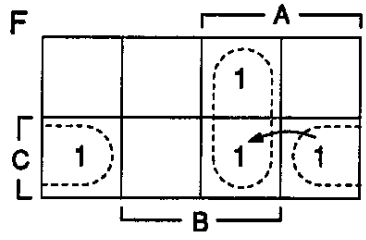
Note 1: Si les 2 "NAND" sont identiques
$\Delta T_{GLITCH} = \Delta T_1$ (car $\Delta T_2 = \Delta T_0$)

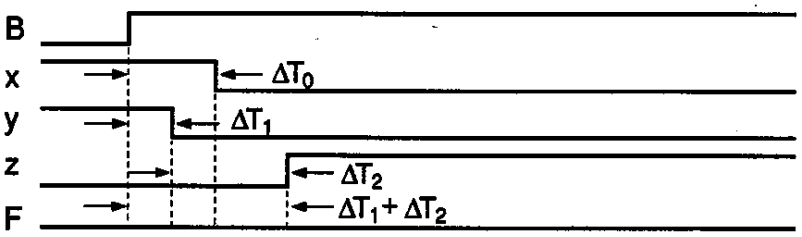
Note 1: Si par hazard $\Delta T_0 = \Delta T_1 + \Delta T_2$ alors pas d'aléa!

La transition momentanée à "0" est un aléa.
Sa durée :

$$\tau = \underbrace{\Delta T_1 + \Delta T_2 + \Delta T_3}_{\text{temps de propagation de B par la voie inférieure}} - \underbrace{(\Delta T_0 + \Delta T_3)}_{\text{temps de propagation de B par la voie supérieure}}$$

On peut démontrer que l'aléa se produit seulement dans un sens de la transition, sens dépendant de la voie de transmission la plus lente. Ainsi, si on garde A = C = 1 mais que B passe de "0" à "1" on a :





F n'a aucun aléa!

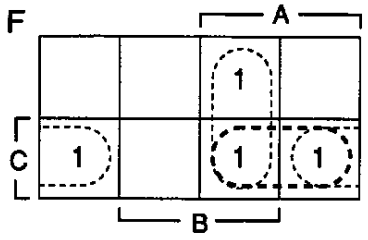
3.14.3 - Correction des aléas

Bien que les circuits synchrones n'exigent pas la correction des aléas, ce n'est pas le cas des circuits asynchrones ou de toute autre forme de circuit dans lesquelles un régime transitoire d'une des fonctions internes modifierait le comportement du circuit. Il faut, dans ces cas, éliminer tout risque d'aléa.

La première règle est d'amortir tout interrupteur susceptible de causer des aléas dynamiques qui pourraient avoir un effet sur le circuit.

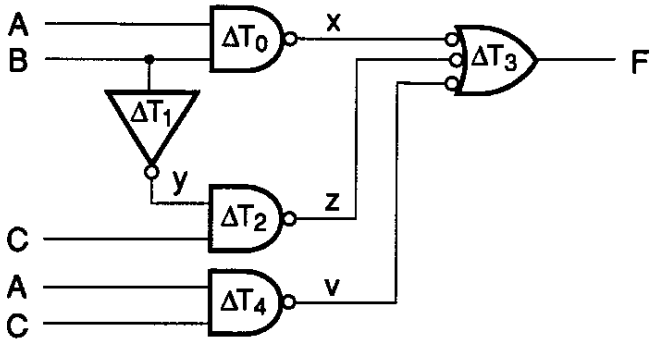
La seconde règle est d'ajouter des termes redondants qui lient tous les groupements séparés par une seule transition de v.i. dans la table de Karnaugh. Ceci s'explique par le fait qu'un aléa statique est dû à une transition d'un groupement à un autre. Cette règle s'applique pour des réalisation de type "POS" ou "SOP" : Il faut faire attention lorsqu'on décide de factoriser par la suite.

Exemple :



$F = AB + \bar{B}C + AC$

terme redondant qui élimine le risque d'aléa

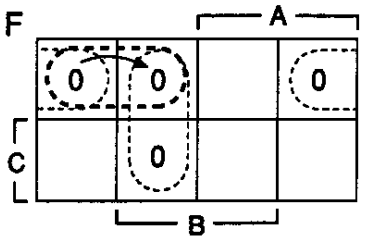


A = C = 1
B passe de "1" à "0"

Comme v = AC et que A = C = 1 ne changent pas alors v = 0 en tout temps ce qui permet à "F" de demeurer à "1" malgré les transitions de "x" ou "y". Aucun aléa.

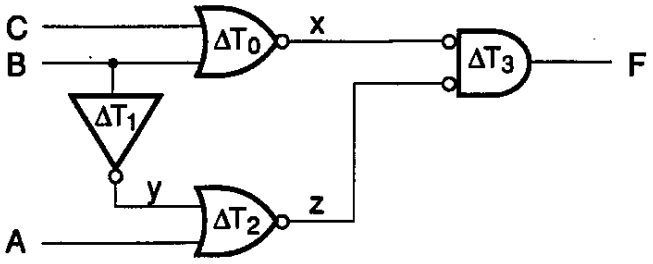
Exemple : F = f (A,B,C)

$F = \Pi M\ (0,2,3,4)$



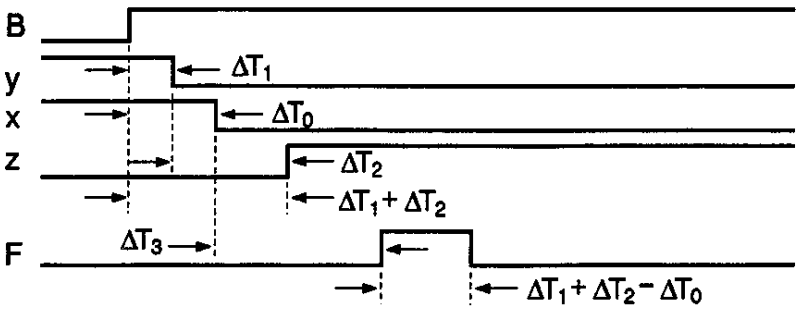
$F = \bar{B}\bar{C} + \bar{A}B$

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



A = C = 0

B passe de "0" à "1"



Par contre si $F = (B+C)(A+\bar{B})(A+C)$

F demeure à "0" car le terme $(A+C) = 0$

On démontre de plus, que l'aléa statique est un "0" transitoire dans la réalisation de "F" par une somme de produits ; un "1" transitoire dans la réalisation de "F" par un produit de sommes.

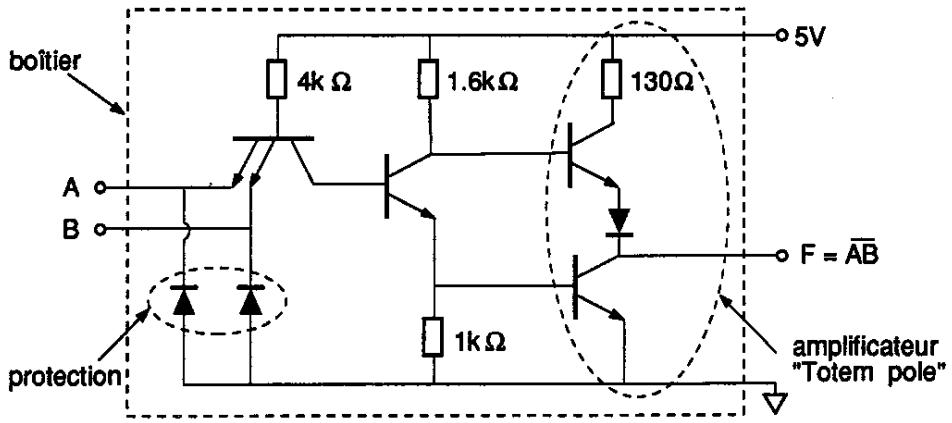
3.15 - Les fonctions spéciales

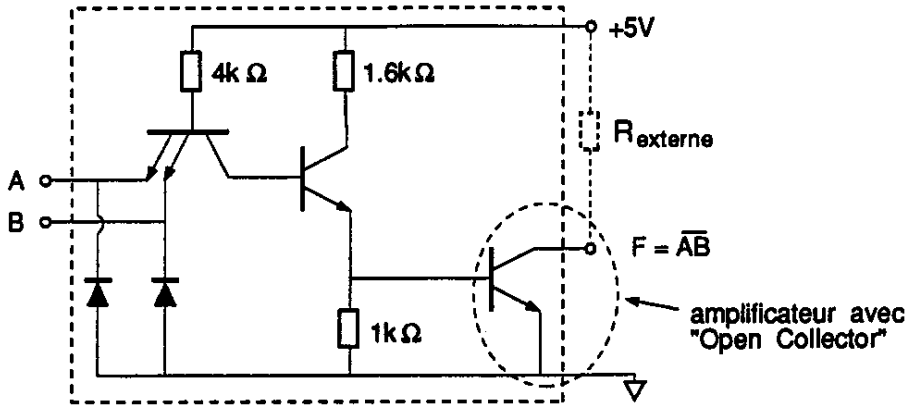
3.15.1 - Le "ET câblé" ("wired and")

Dans certains cas, les sorties de plusieurs portes spéciales peuvent être raccordées ensemble pour créer une fonction logique câblée: c'est une porte câblée dont ses entrées ne sont ni plus, ni moins que les sorties des portes spéciales et sa sortie est produite par le raccordement.

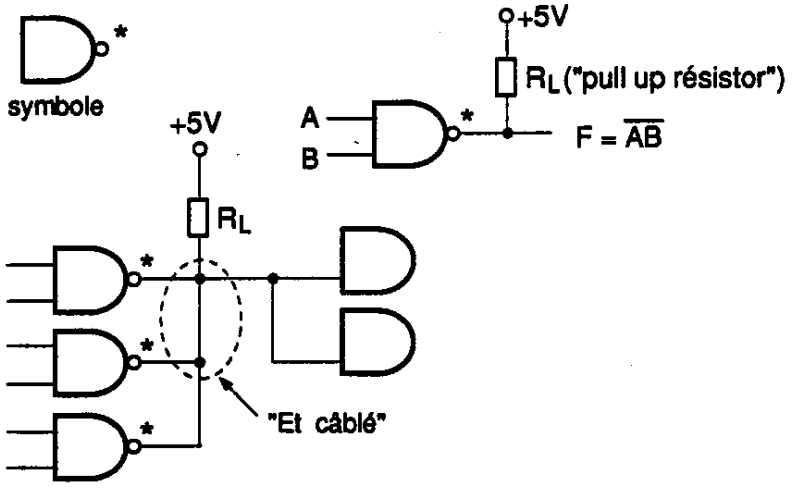
La figure ci-dessous donne le circuit électronique d'une porte NAND TTL à 2 entrées. Dans cette porte, la logique AND est faite par le transistor à émetteurs multiples (pour une porte NOR, la logique est réalisée par des transistors en parallèle qui agissent comme des contacts en parallèle). L'amplificateur inverseur de sortie, en TTL, peut être du type "Totem pole" ou "Open collector".

amplificateur de sortie en TTL → "Totem pole"
→ "Open collector"





L'amplificateur "Totem Pole" est l'amplificateur standard mais l'amplificateur avec avec "Open Collector" sert partout où il y a des charges difficiles à entraîner ou encore pour réaliser des interfaces entre TTL et une famille logique ou encore pour réaliser les portes "ET Câblé". Le principe est de relier les sorties mettant ainsi les transistors de sortie en parallèle. Il faut ensuite brancher la résistance extérieure ("Pull up resistor"), résistance qui est calculée en fonction du nombre de portes à collecteur ouvert connectées, ainsi que du nombre et du type d'entrées TTL qui y sont branchées.



$$R \leq \frac{V_{cc} - V_{OUT}(1)}{n\,I_{OUT}(1) + m\,I_{IN}(1)}$$

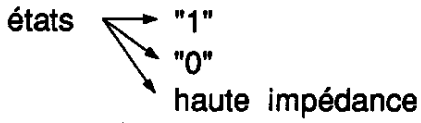
$$R \geq \frac{V_{cc} - V_{OUT}(0)}{n\,I_{OUT}(0) + m\,I_{IN}(0)}$$

avec n = nombre de sorties rattachées
m = nombre d'entrées branchées

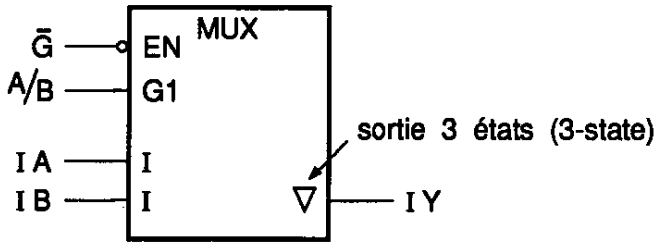
$I_{OUT}(0)$ ~ 15 mA
$I_{OUT}(1)$ ~ − 250 µA
$I_{IN}(0)$ ~ −1.6 mA
$I_{IN}(1)$ ~ 40 µA
$V_{OUT}(0)$ ~ 0-0.4 V
$V_{OUT}(1)$ ~ 2.4-5 V

3.15.2 - Sortie 3 états

Une sortie 3 états, comme son nom l'indique possède 3 états.



La sortie haute impédance permet à la porte ou à la fonction logique de devenir transparente dans le circuit. C'est un peu comme si la porte ou la fonction logique n'existait plus. En effet, une sortie haute impédance peut se voir comme un fil laissé dans les airs. L'impédance au bout du fil est infinie et le fil, branché nulle part, ne modifie en rien le comportement du reste du circuit (en haute fréquence, il risque d'agir comme antenne et capter du bruit mais ce n'est pas la préoccupation ici). Ainsi, la porte ou la fonction logique a un signal de commande qui permet à celle-ci de réagir normalement ou de devenir transparente dans le circuit. Ce contrôle est le signal d'inhibition.



3.15.3 - Entrées inutilisées

Il n'est pas recommandé de laisser flottante une entrée inutilisée car on augmente la susceptibilité au bruit du système et on risque d'allonger le temps de propagation du circuit (en CMOS, on peut même détruire le circuit intégré).

En TTL, les entrées inutilisées devraient être connectées soit à la référence ou au niveau logique "I" selon la fonction la connexion à "I" puisqu'elle impose une pénalité beaucoup plus faible sur l'alimentation. En effet, une entrée non branchée est au niveau logique "I". Attention cependant aux surtensions de l'alimentation car les entrées sont moins tolérantes à une surtension (5.5 V maximum) que l'alimentation du circuit intégré (7 V maximum).

CHAPITRE 4

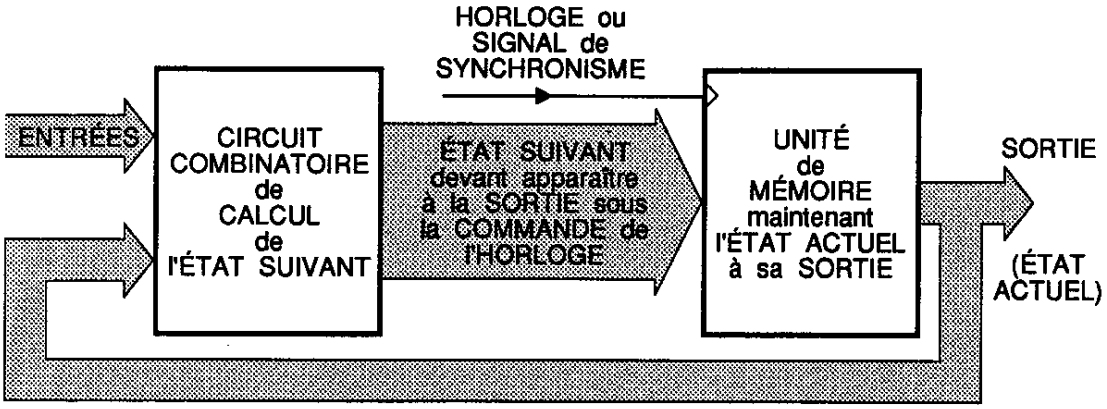
ÉLÉMENTS SYNCHRONES
(Bascules, régistres et compteurs)

INTRODUCTION

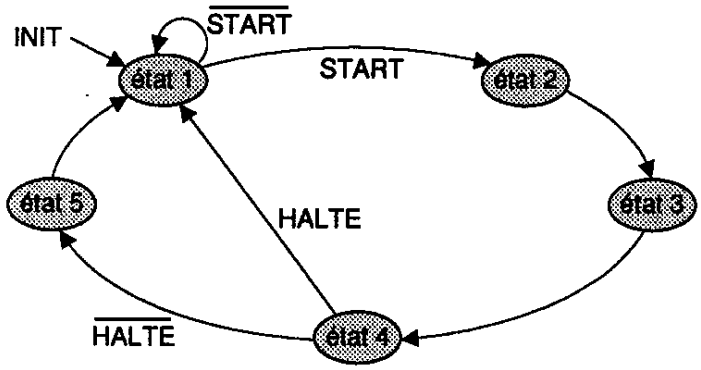
Jusqu'à maintenant, on a vu que des circuits combinatoires, c'est-à-dire des circuits où les sorties à un instant donné ne dépendent que des entrées, or, il arrive souvent qu'on désire concevoir un circuit qui possède un nombre "FINI" et "DÉTERMINÉ" d'ÉTATS. De plus, lorsque le circuit se trouve dans un état donné, l'état qui le suit (dans le temps) ne peut être quelconque, mais dépend plutôt de l'état actuel "ET" des entrées présentes. On appelle un tel circuit une "MACHINE À ÉTATS FINIS" ou machine séquentielle.

Le fait que la suite temporelle des états ne peut être quelconque implique qu'il faudra utiliser les sorties comme entrées pour le calcul de l'état suivant. De plus, le passage d'un état donné à l'état suivant doit se faire à un moment précis, d'où la nécessité d'un signal de synchronisation commun à tous les éléments "étatiques" du système.

On peut donc représenter un circuit séquentiel par le schéma suivant :



L'état actuel change sous la commande du signal de synchronisation (ou horloge) et devient un certain état suivant dépendamment des niveaux des entrées, de l'état actuel. Un circuit séquentiel se comporte de façon telle qu'on représente son fonctionnement par un diagramme d'états.



4.1 - Concept de mémorisation

Un unité de mémoire est un circuit qui permet d'emmagasiner une formation tant et aussi longtemps qu'on le demande. Cette information est disponible en tout temps à la sortie. Une ou plusieurs commandes procurent à l'unité la capacité de modifier son contenu ou de le sauvegarder. L'information enregistrée s'appelle la donnée ("data").

Il y a fondamentalement, 2 classes d'unité mémoire. La première effectue chargement et modification dans les intants immédiats suivants la demande. La seconde agit sur l'ordre d'une commande extérieure de synchronisation communément appelée "horloge" ("H"). C'est l'horloge qui dicte le temps. La cellule binaire est la seule représentante de la première classe; la seconde regroupe un ensemble de circuits qu'on appelle les bascules.

unité mémoire
- cellule S-R (sans horloge)
- bascules (avec horloge)

4.2 - La cellule binaire S-R

La cellule binaire S-R ("set-reset") constitue l'élément fondamental de la logique séquentielle : c'est la mémoire unitaire (un bit) la plus simple sur laquelle repose la conception des bascules. Elle est, de plus, l'élément de base des mémoires vives (RAM = "random access memory") à semi-conducteurs. (On parle ici de mémoire "Statique")

La cellule binaire note la présence fugitive d'une information et la conserve. Elle possède 2 entrées de commande ("set" et "reset") et une sortie Q.

La sortie "Q" demeure inchangée lorsque $S = R = 0$; va obligatoirement à 0 lorsque $S = 0$ et $R = 1$; et va obligatoirement à "1" lorsque $S = 1$ et $R = 0$. La combinaison on $S = R = 1$ est considérée comme illogique de sorte qu'elle ne devrait pas se produire.

S = "set" ⟶ mise à "1"
R = "reset" ⟶ mise à "0"

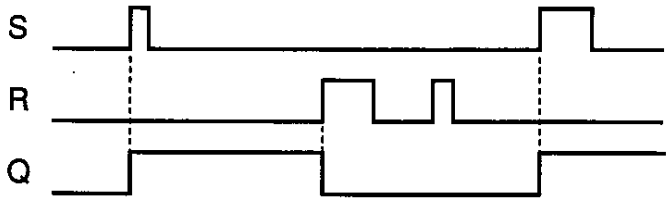
S	R	Q_n	Q_{n+1}	
0	0	0	0	⇄ *la sortie reste inchangée*
0	0	1	1	
0	1	X	0	⟵ *mise à "0"*
1	0	X	1	⟵ *mise à "1"*
1	1	X	?	⟵ *ne pas faire, résultat imprévisible*

avec

Q_n = *valeur de la sortie précédant la commande*
Q_{n+1} = *valeur de la sortie après l'action transitoire ou suivante*

diagramme de fonctionnement *

Q initialement à "0"

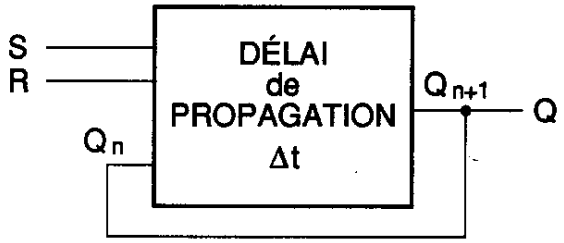


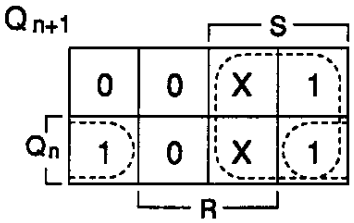
* délai de propagation non considéré

D'après la table de vérité, la nouvelle valeur de "Q" dépend de celle qu'elle avait précédemment. En conséquence de quoi, on peut conclure que la sortie "Q" agit comme une entrée (ou v.i.) au même circuit. C'est la récursivité ("feedback").

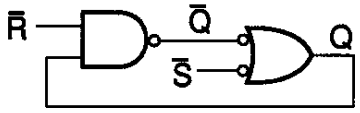
Réalisation :

On décompose le problème de la synthèse de la cellule binaire en le regardant sous cet angle :

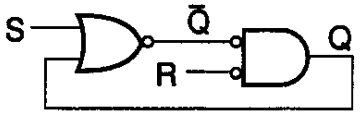




SOP	POS
$Q_{n+1} = \bar{R}Q_n + S$	$\bar{Q}_{n+1} = (R + \bar{Q}_n) \cdot \bar{S}$
$\bar{Q}_{n+1} = \bar{S} \cdot \bar{Q}_n + R$	$Q_{n+1} = (S + Q_n) \cdot \bar{R}$



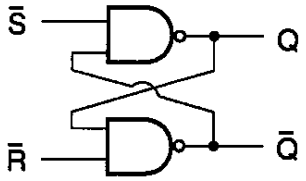
réalisation avec des portes "NAND"



réalisation avec des portes "NOR"

Le délai de propagation est en fait le double de celui d'une porte "NAND" ou "NOR".

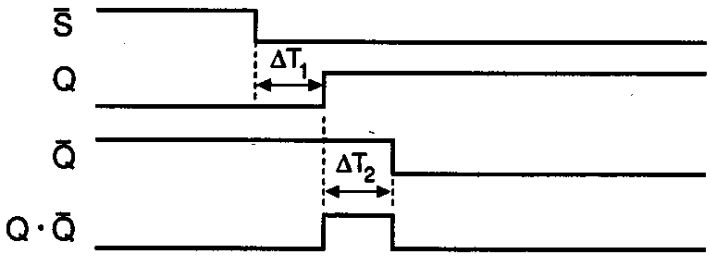
Habituellement, on préfère dessiner le circuit de la cellule binaire d'une autre façon équivalente. Il devient plus facile alors de vérifier le comportement du circuit face aux commandes "S" et "R". C'est d'ailleurs de cette façon qu'on l'aperçoit dans les data book (TTL 74279).



Problèmes dynamiques

La grande majorité des problèmes inhérents à la récursivité est due à la méconnaissance de principe de base de cette structure appuyée sur les délais de propagation. En outre, selon le schéma de la cellule binaire, il est virtuellement impossible que le produit "$Q \cdot \bar{Q}$" soit nul en tout temps!

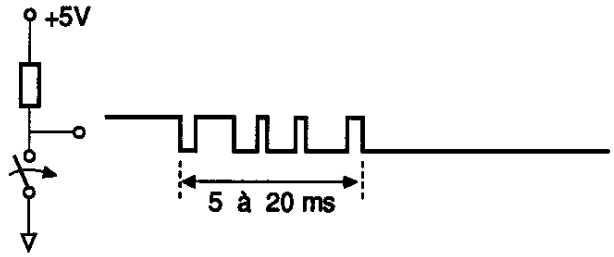
Exemple : $\bar{R} = 1$
Q initialement à "0"
$\bar{S}$ passe de "1" à "0"



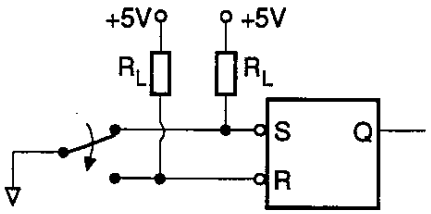
Bien plus, la table de vérité ne révèle pas que certains états sont transitoires et ne durent que l'instant du délai de propagation. Par exemple, l'état ou $S = 1$, $R = Q_n = 0$ est instable (qui dure ΔT_1) puisque l'état stable vers lequel il tend est $S = 1$, $R = 0$ et $Q_n = 1$. Ainsi une commande trop brève (une impulsion de durée inférieure au temps de propagation de la cellule) risque de faire revenir la cellule dans son état initiale après qu'ait commencé la transition.

Exemple d'utilisation de la cellule binaire
L'amortissement de contacts mécaniques ("debouncing") :

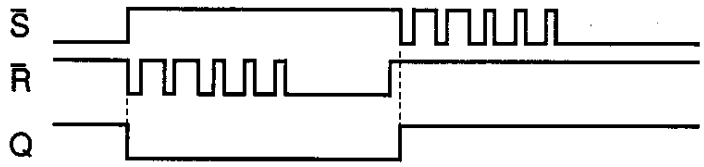
Un interrupteur est une pièce mécanique qui a un comportement fâcheux lorsqu'on l'ouvre ou le ferme comme l'illustre la figure ci-dessous.



La règle d'or consiste à toujours amortir les interrupteurs sauf pour des entrées statiques où là, la règle n'est plus essentielle. Pour ce faire, il s'agit de prendre une cellule binaire avec un interrupteur à 2 positions.

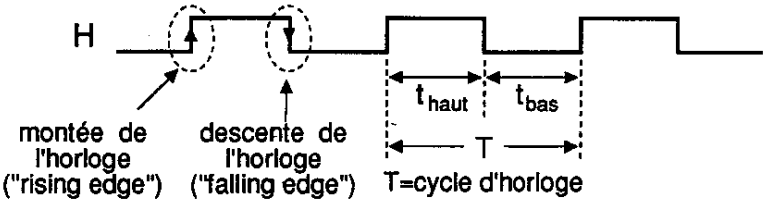


$$0 < R_L < \frac{V_{cc} - V_{out}(1)}{I_{in}(1)}$$



4.3 - Le signal d'horloge

Le signal d'horloge sert à synchroniser l'action pour la rendre discrète dans le temps. C'est une onde carrée produite par un oscillateur (triggers de Schmitt ou amplificateur avec hystérésis auquel on insère de la rétroaction).



$f = 1/T$

f = fréquence de l'horloge en Hertz (T en secondes)

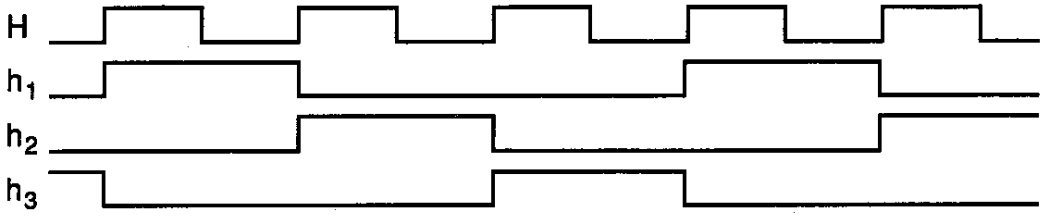
$$\text{"duty cycle"} = \frac{t_{haut}}{t_{haut} + t_{bas}} = \frac{t_{haut}}{T}$$

En tant que commande de synchronisation, l'horloge est présente dans tous les systèmes séquentiels synchrones par opposition à ceux asynchrones. À ce titre,

> ⟶ L'horloge doit se rendre directement à toutes les entrées de synchronisme (notée ⊢▷) des éléments qui composent le système séquentiel synchrone.

On peut se demander "comment opère le mécanisme de synchronisation avec l'horloge?". À cela, on répond que la synchronisation d'un circuits consiste à produire une action sur l'ordre d'une commande. Il en est de même pour un circuit logique où l'action, cette fois, s'effectue uniquement à la montée (plus généralement) ou à la descente de l'horloge (mais non les 2 à la fois dans un même circuit). Pour un élément mémoire, par exemple, le chargement de la donnée se fait au moment de la montée de l'horloge ; c'est la valeur présente à ce moment qui est sauvegardée.

Dans un système séquentiel complexe, où l'action se déroule en plusieurs étapes, on subdivise l'horloge (qui s'appelle alors l'horloge mère) en plusieurs phases (h_i). Ces phases établissent la chronologie entre les actions mais encore là, l'horloge mère continue à se rendre directement aux entrées de synchronisme ; les phases agissant sur les autres commandes. La figure ci-dessous montre la subdivision de l'horloge mère en 3 phases.



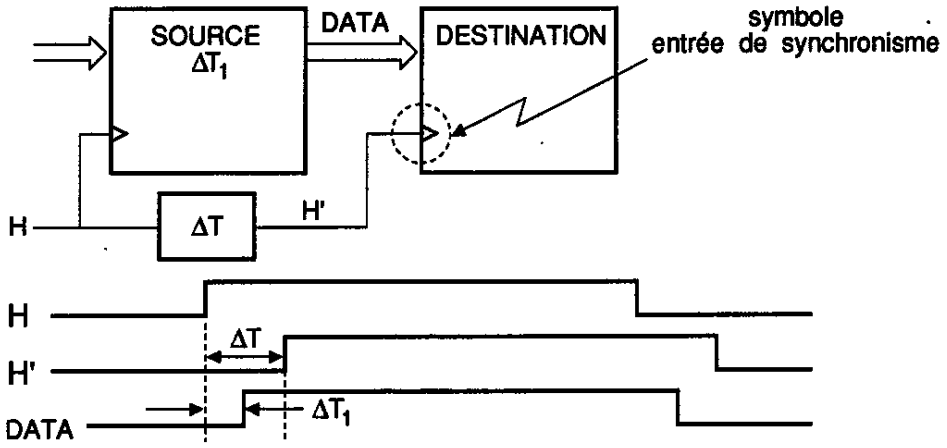
Ainsi, dans un système synchrone, une même horloge contrôle une batterie d'éléments. En pratique, dans un gros système, il est possible d'avoir des décalages entre les entrées de synchronisme des éléments qui sont difficiles à éviter.

* *Le décalage peut provenir de l'utilisation de plusieurs "buffers" en parallèle qui distribuent l'horloge sans dépasser les limites de charge imposées par le "fan-out". Il y a avantage à prendre les "buffers" venant d'un même boitier.*

* *Les délais de propagation dans le câblage (~1-1.5 ns / pied) peuvent causer des décalages d'horloge importants surtout dans une famille logique.*

Ces décalages ("clock skew") sont possiblement la cause d'aléas de fonctionnement dans un circuit synchrone car au moment du chargement, il est possible que la donnée ait changé à cause du décalage entre les commandes synchrones de la source et de la destination. La solution au problème consiste à ne pas modifier une variable au moment où l'on s'en sert (d'ajouter des étapes dans le processus ou dans l'organigramme d'un séquenceur si nécessaire). On n'a pas à appliquer cette règle pour des petits systèmes.

Exemple : "clock skew"



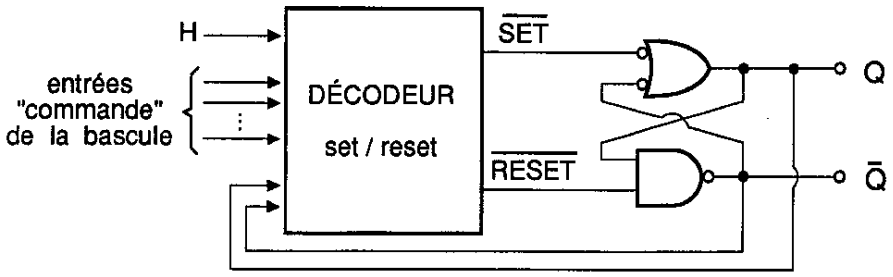
la destination enregistre DATA = 1 plutôt que DATA = 0.

4.4 - Bascules élémentaires

En ajoutant un circuit de contrôle aux entrées d'une cellule binaire "S-R", il est possible de créer une bascule élémentaire.

4.4.1 - Modèle général

Le modèle général des bascules élémentaires apparait à la figure ci-dessous.



Le décodeur "set / reset" active l'une des lignes "$\overline{SET}$" ou "$\overline{RESET}$" (active bas) à partir des entrées "commande" et de l'état présent disponible aux sorties "Q" et $\bar{Q}$ branchées en rétroaction. L'horloge synchronise l'action de la bascule.

Pour faire la synthèse du décodeur "set / reset", il suffit de répondre à 2 questions pour chaque condition d'entrée.

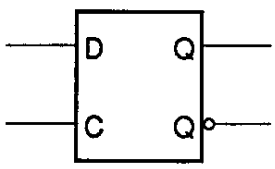
1 : doit-on activer le signal "$\overline{SET}$" de la cellule binaire?
2 : doit-on activer le signal "$\overline{RESET}$"?

Cette synthèse du décodeur par le jeu des questions se fait à partir des tables caractéristique et d'excitation propre à la bascule. La table caractéristique n'est, qu'une table de vérité spéciale qui spécifie la caractéristique opérationnelle de la bascule. La sortie Q_{n+1} de la bascule est ici fonction des entrées "commande" et de l'état présent Q_n. La table d'excitation montre quelles sont les conditions d'entrée pour réaliser une transition. Ces tables font appel aux états Q_n et Q_{n+1} pour montrer le principe séquentiel des bascules.

4.4.2 - La bascule "D" ("latch D")

La bascule "D" (D pour "delay") enregistre directement la valeur présentée à l'entrée "D" et la transfère à la sortie "Q", sous la commande de l'horloge (i.e. à un moment précis de celle-ci).

Symbole :



(TTL 7475, 74100, 74116)

Table caractéristique :

D	Q_n	Q_{n+1}	opération sur les entrées SR
0	0	0	X RESET (SET=0)
0	1	0	RESET (SET=0)
1	0	1	SET (RESET=0)
1	1	1	X SET (RESET=0)

Table d'excitation :

$Q_n \rightarrow$	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

On déduit d'après la table de Karnaugh que

$$RESET = \bar{D}$$
$$SET = D$$

Mais pour tenir compte de la commande de synchronisme, dans une bascule élémentaire, on multiplie la commande au préalable par C.

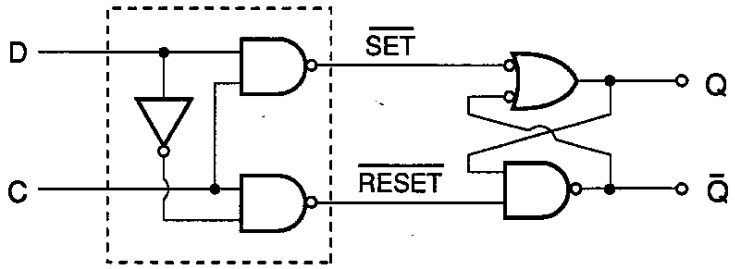
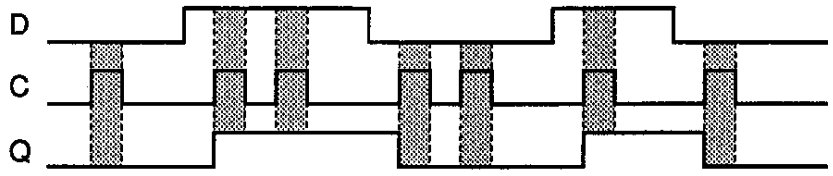


Diagramme de fonctionnement :

Q initialement à 0



4.4.3 - La bascule J - K

La bascule "J - K" peut être vue comme une bascule "S-R" avec horloge où il est possible d'activer les 2 commandes "J" et "K" à la fois (on sait que "S = R = 1" est à éviter). De cette façon,

J → SET
K → RESET
Si J=K=1 alors la bascule renverse le $Q_{n+1} = \bar{Q}_n$ ("toggle")

Table caractéristique :

J	K	Q_n	Q_{n+1}	opération
0	0	0	0	X RESET (SET = 0)
0	0	1	1	X SET (RESET = 0)
0	1	0	0	X RESET (SET = 0)
0	1	1	0	RESET (SET = 0)
1	0	0	1	SET (RESET = 0)
1	0	1	1	X SET (RESET = 0)
1	1	0	1	SET (RESET = 0)
1	1	1	0	RESET (SET = 0)

Table d'excitation :

$Q_n \rightarrow$	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

à partir de la table caractéristique, on remplit les tables de Karnaugh, une pour "SET" et l'autre pour "RESET".

SET

Q_n \ J	0	0	1	1
0	X	0	0	X

K

RESET

Q_n \ J	X	X	0	0
0	0	1	1	0

K

$SET = J \cdot \bar{Q}_n \cdot C$

$RESET = K \cdot Q_n \cdot C$

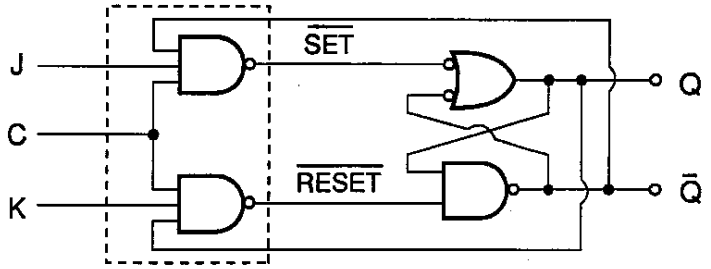
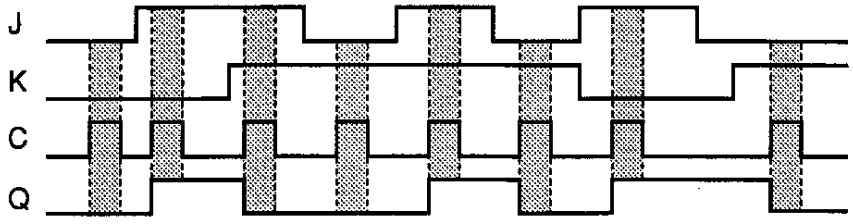


Diagramme de fonctionnement :

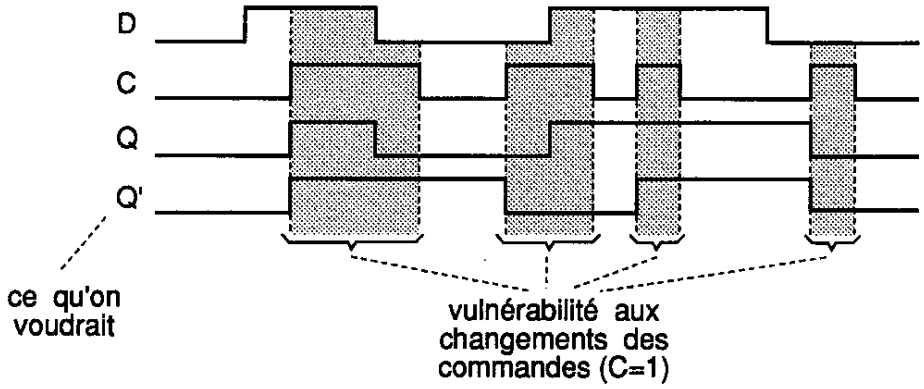
Q initialement à 0



4.4.4 - Problème des bascules élémentaires

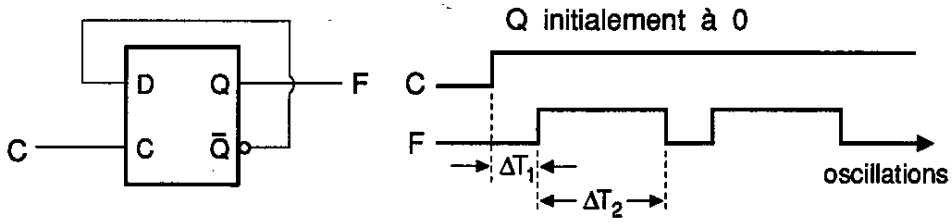
Les bascules élémentaires sont vulnérables aux changements des commandes d'entrée lorsque l'horloge est au niveau logique haut (C=1). En effet, lorsque C=1, tout changement des commandes se répercute à la sortie. Ceci est contraire à la philosophie du synchronisme avec l'horloge qui exige que toute action s'effectue à la montée ou à la descente seulement du signal d'horloge.

Exemple : avec "latch D"



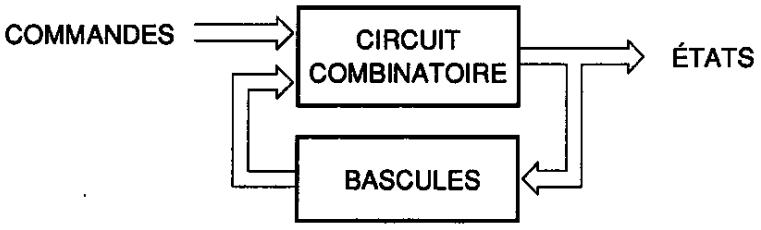
Cette lacune cause des problèmes d'aléas dans les circuits séquentiels et provoque des oscillations lorsqu'on utilise ces bascules avec rétroaction de la sortie sur les commandes.

Exemple :



T_1 = délai de propagation de la bascule

T_2 = délai de propagation du circuit combinatoire dans le modèle du circuit séquentiel suivant :



La solution semble être dans l'utilisation d'une 2ième couche de bascules qui serait opérée dans l'autre phase de l'horloge. Ce qui implique 2 étages de bascules élémentaires dont le premier, appelé maître, enregistre les commandes dans un 1er temps, tandis que le second, l'esclave, mémorise et fait l'action de la donnée du maître dans un dernier temps. C'est la bascule maître-esclave.

4.5 - Bascule "J - K" maître esclave

(ex : 74106 : negative edge triggered J-K)
(ex : 74107 : positive edge triggered J-$\bar{K}$)

4.5.1 - Principe

La bascule "J-R" maître-esclave est formée de cellules "S-R" avec commandes de validation C_1 et C_2 respectivement telles que C_1 et C_2 sont disjointes ($C_1 \cdot C_2 = 0$ en tout temps). Les cellules sont placées en cascade, l'une commandée par C_1 et l'autre par C_2 .

Le premier étage prend la décision lorsque C_1 est actif et le second exécute l'action lorsque C_2 est actif. On obtient alors la réalisation suivante :

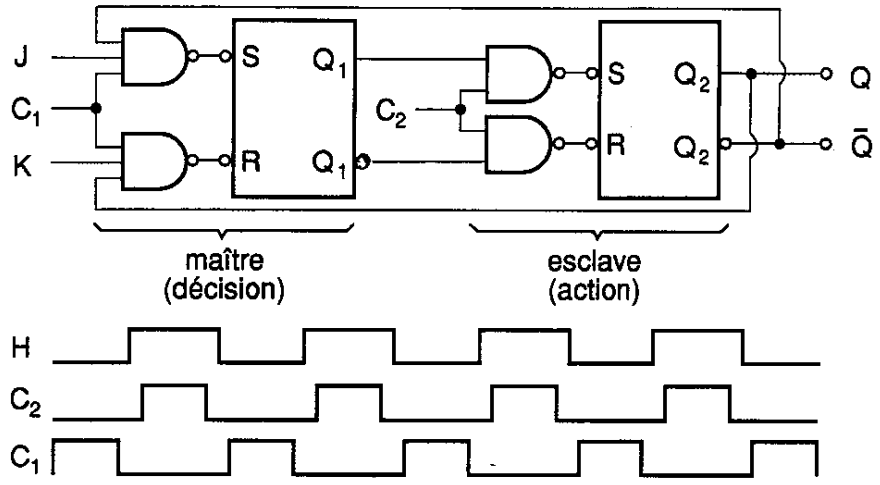
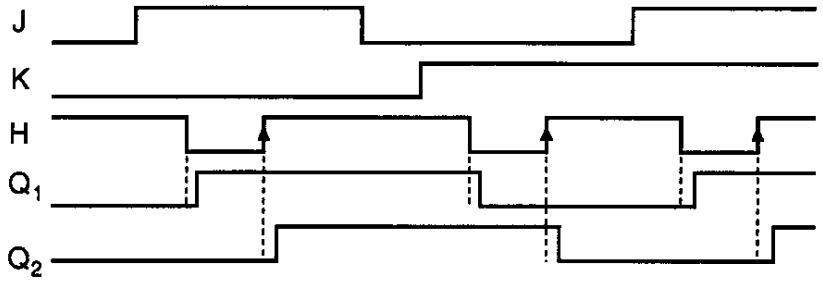


Diagramme de fonctionnement :

Q_1 et Q_2 initialement à 0



Le diagramme de fonctionnement se vérifie en constatant que :

- Si H=1, Q_1 ne change pas ($Q_{1\,n+1} = Q_{1\,n}$)
  tandis que Q_2 prend la valeur de Q_1 ($Q_{2\,n+1} = Q_{1\,n}$)
- Si H=0, Q_2 ne change pas ($Q_{2\,n+1} = Q_{2\,n}$)
  tandis que Q_1 devient

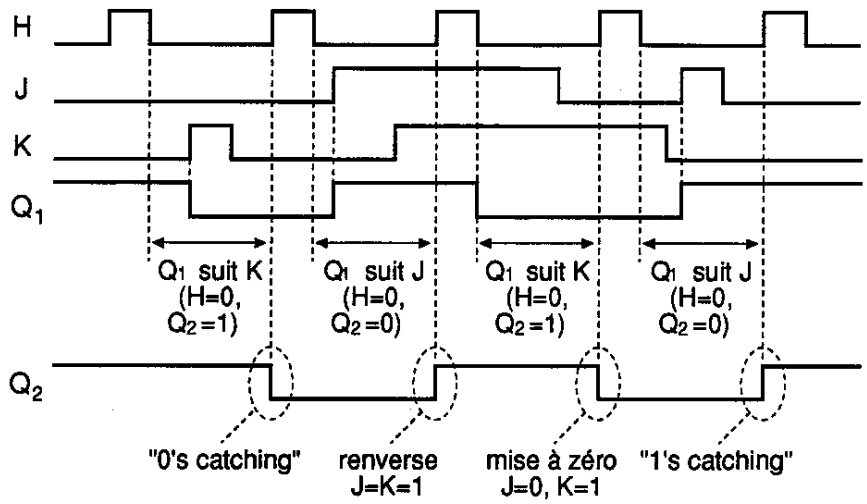
$$Q_{1\,n+1} = J_n \cdot \overline{Q_{2\,n}} + \overline{K_n \cdot Q_{2\,n}} \cdot Q_{1\,n}$$

Vous pouvez vérifier que la dernière équation suit la table d'excitation d'une bascule "J-K" élémentaire.

4.5.2 - "1 et 0 catching"

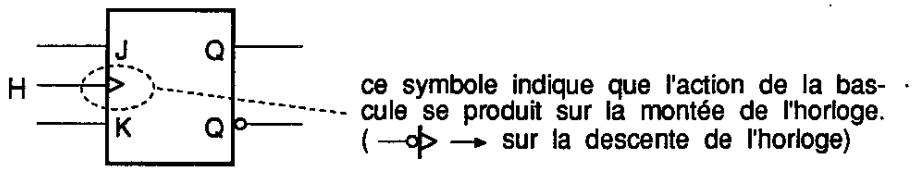
La bascule "J-K" maître-esclave a un comportement indésirable lorsque les commandes J et K varient pendant que le l'horloge est au niveau logique bas (H=0). Ce comportement s'appelle le "one-catching" et le "zero-catching". C'est en fait un problème puisque alors la bascule ne répond plus au synchronisme selon la table d'excitation d'une bascule J-K.

Exemple :



4.5.3 - Symbole avec synchronisme

Malgré le problème rattaché au principe même de la bascule maître-esclave ("master-slave"), on inclut la bascule maître-esclave dans les rangs des circuits synchrones i.e. en synchronisme avec la montée ou la descente de l'horloge. C'est pour cette raison qu'on insère le symbole " ⊣▷ " identifiant un élément synchrone. Il faut toutefois se rappeler du problème et le contourner en ne modifiant pas les commandes de la bascule maître-esclave lorsque l'horloge est basse.



4.6 - Les bascules "edge-triggered"

Les bascules "edge-triggered" sont les bascules synchrones par excellence. Ils échantillonnent les commandes et s'actionnent en synchronisme avec l'horloge tel que stipulé théoriquement.

4.6.1 - D "edge-triggered"

La bascule D "edge-triggered" (ou D tout simplement) se réalise, à la base, en prenant 2 "latches D" en cascade ou les commandes de validation de chacune sont disjointes.

La première "latch D" enregistre et transfère la donnée présente à l'entrée D. Sa sortie suit l'entrée lorsque H = 0. Pendant ce temps la seconde "latch D" est maintenue constante. Lorsque H = 1, la première "latch D" reste bloquée ("data lock out") et la seconde transfère l'information enregistrée par la première.

→ Les changements des commandes n'ont aucun effet lorsque la sortie change dans une bascule "edge-triggered".

Réalisation :

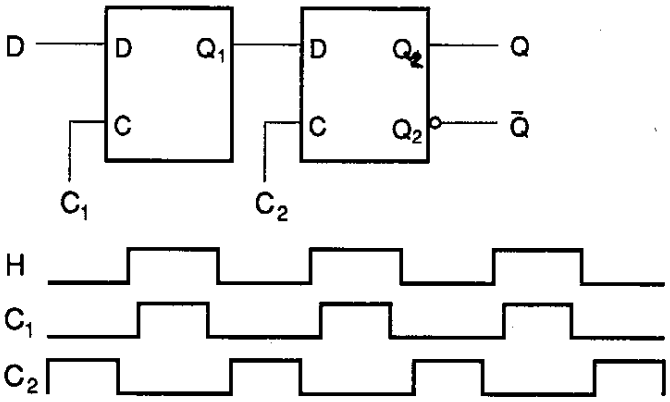
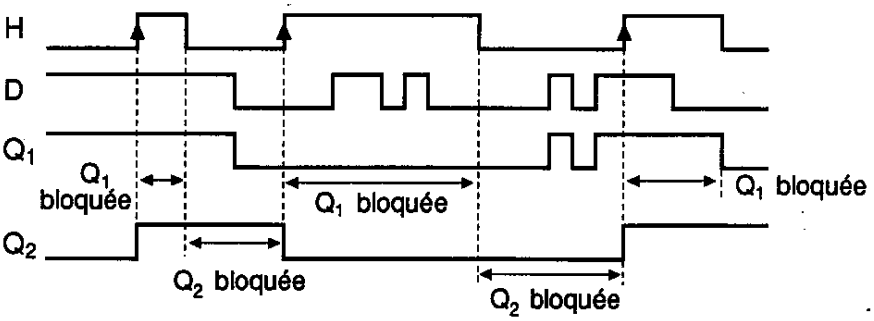
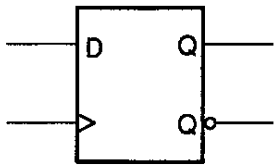


Diagramme de fonctionnement :



Symbole :



(TTL 7474, 74174, 74175)

4.6.2 - J - K "edge-triggered"

La bascule J - K "edge triggered" (ou J - K tout simplement) réagit comme la bascule J - K maître-esclave dans son fonctionnement normal. La bascule se sert d'une bascule D "edge-triggered" pour être elle-même "edge-triggered". C'est, bien sûr, le principe de base permettant la synthèse d'une telle bascule.

La table caractéristique de la bascule "J-K" montre que l'équation caractéristique de la bascule est :

$$Q_{n+1} = J_n \cdot \bar{Q}_n + \bar{K}_n \cdot Q_n$$

Q_{n+1} \ J	0	0	1	1
Q_n = 0	0	0	1	1
Q_n = 1	1	0	0	1

Réalisation de principe :

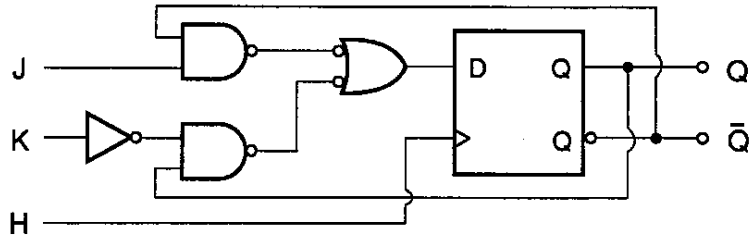
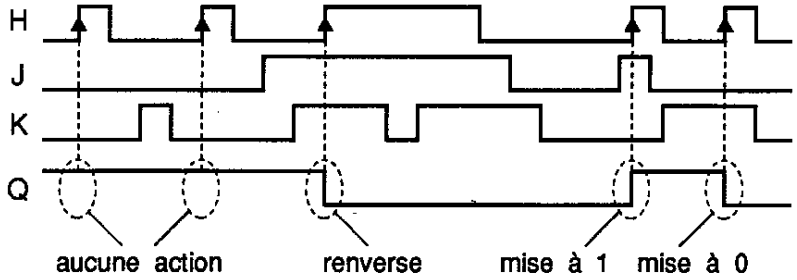


Diagramme de fonctionnement :

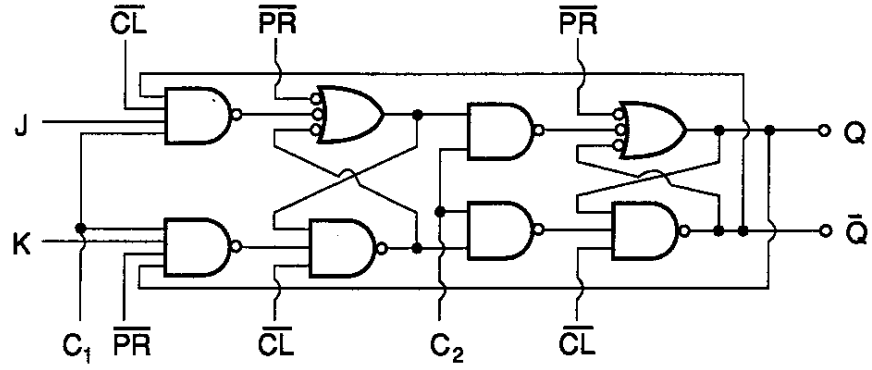


4.7 - Les entrées asynchrones

Les entrées asynchrones sont des commandes supplémentaires rencontrées sur plusieurs boitiers de bascules dont l'effet est immédiat. Ce sont les commandes forcées de mise à 0 ($\overline{CL}$) et de mise à 1 ($\overline{PR}$) à logique négative (active bas).

Elles agissent directement sur les cellules binaires qui composent la bascule. Évidemment, vue la combinaison illogique d'activation simultanée de S et R, les commandes asynchrones ne doivent pas être actives en même temps.

Exemple : entrées asynchrones sur la bascule "J - K" maître-esclave.

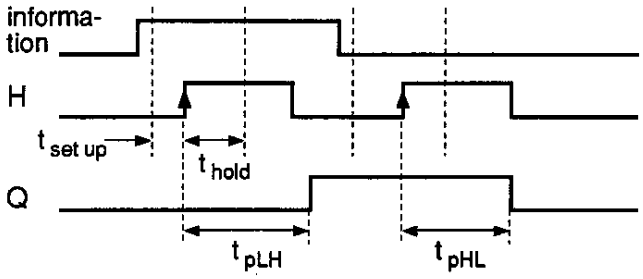


4.8 - Paramètres dynamiques d'une bascule

Le constructeur d'une cellule ou d'une bascule définit un certain nombre de paramètres que l'utilisateur doit respecter pour obtenir un fonctionnement correcte dans le temps. Ce sont :

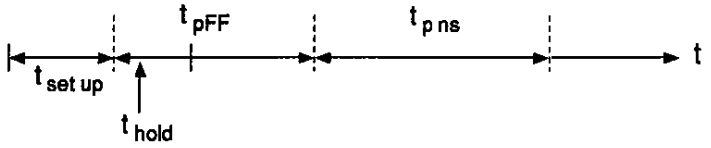
* *Le temps de préaffichage ("set up time"): l'information présente sur les entrées dépendant de l'horloge doit être affichée un certain temps avant le front défini de l'horloge.*

* *Le temps de maintien ("hold time"): l'information présente sur les entrées dépendant de l'horloge doit rester affichée un certain temps après le front défini de l'horloge.*



* temps de renversement d'une sortie initialement à "0" pour passer au niveau "1" (t_{PLH})
* temps de renversement d'une sortie initialement à "1" pour passer au niveau "0" (t_{PHL})
* durée minimum des impulsions de commande
* fréquence maximale de l'horloge.

  Exemple : dans le modèle du circuit séquentiel représenté au paragraphe 4.4.4, on trouve :



$$F_{max} \leq \frac{1}{t_{set\ up} + t_{pFF} + t_{p\,ns}}$$

où t_{pFF} = temps de propagation des bascules
$t_{p\,ns}$ = temps de propagation du circuit combinatoire

Exemple :

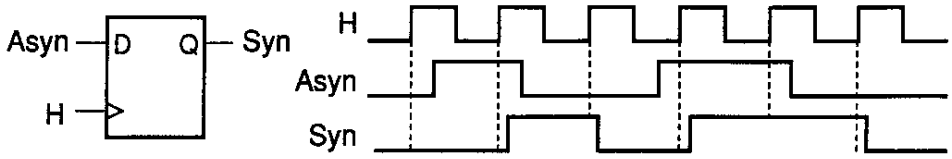
	7474	74109
$t_{set\ up}$	20 ns	10 ns
t_{hold}	5 ns	6 ns
t_{pHL}	20-40 ns	18-28 ns
t_{pLH}	14-25 ns	10-16 ns
f_{max}	25MHz	33MHz

4.8.1 - Analyse de circuits synchrones

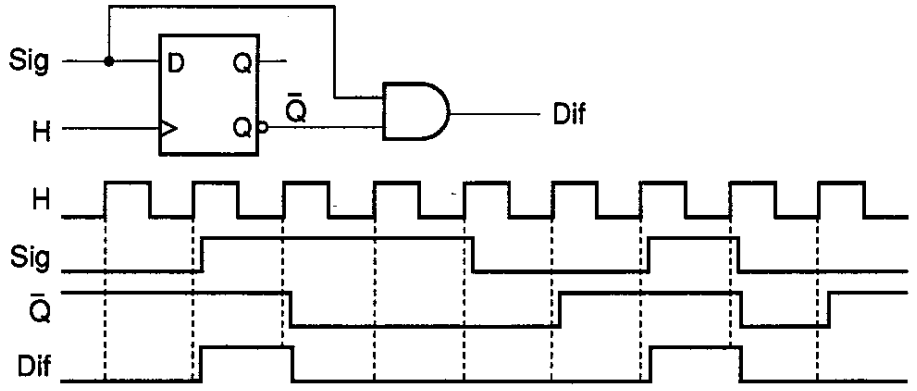
Tout circuit synchrone est composé, à la base, de bascules D et/ou J-K "edge triggered"; l'horloge se rend directement à toutes les entrées de synchronisme (⊢) des éléments qui forment le circuits et tous les signaux utilisés et produits doivent être synchronisés, au préalable, sur l'horloge.

L'analyse d'un circuit synchrone exige une procédure méthodique, ordonnée et séquentielle. Le principe est d'établir à chaque pas, l'état suivant du circuit. Pour ce faire, on utilise les tables caractérisriques des bascules.

Exemple #1 : Synchronisation des signaux externes



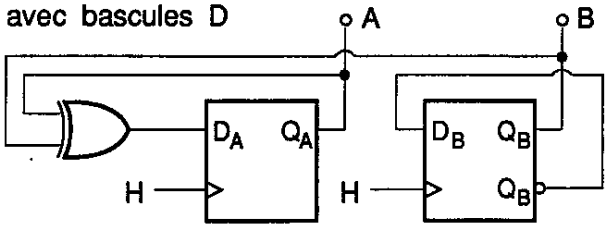
Exemple #2 : Différentiateur d'un front montant



Attention: *Les signaux synchrones sont synchronisés sur et par l'horloge H. Ils sont donc légèrement en retard sur la montée de H, ce retard provenant des délais de propagation des éléments qui ont produit ces signaux.*

Exemple #3 : Compteur 2 bits

avec bascules D



CODE	état présent A	état présent B	D_A	D_B	état suivant A	état suivant B
0	0	0	0	1	0	1
1	0	1	1	0	1	0
2	1	0	1	1	1	1
3	1	1	0	0	0	0

(Exemple #3...) L'état suivant dépend directement des valeurs de D_A et D_B présentes à la prochaine montée de H (voir la table caractéristique de la bascule D).

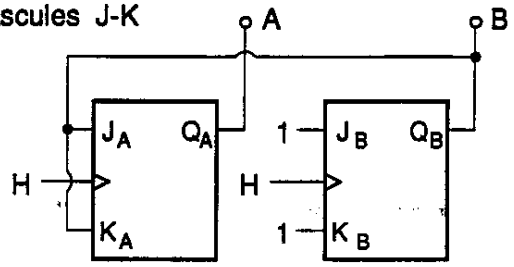
Les valeurs de D_A et D_B sont déterminées à partir de l'état présent.

$D_A = A \oplus B$

$D_B = \bar{B}$

Pour toutes les combinaisons possibles de l'état présent, on vérifie quel est l'état suivant dans le but de connaître la séquence suivie par le circuit.

avec bascules J-K



CODE	état présent A	état présent B	J_A	K_A	J_B	K_B	état suivant A	état suivant B
0	0	0	0	0	1	1	0	1
1	0	1	1	1	1	1	1	0
2	1	0	0	0	1	1	1	1
3	1	1	1	1	1	1	0	0

4.9 - Les registres

4.9.1 - Définition

Un registre est un assemblage de bascules commandées par une horloge commune destiné à mémoriser momentanément une donnée binaire. Cette donnée peut être insérée dans le registre parallèlement ou encore en série. Il est possible ensuite de la décaler et le résultat est disponible en parallèle ou en série.

Les registres sont classés selon le nombre de bits avec lequel ils travaillent et leur type d'opération.

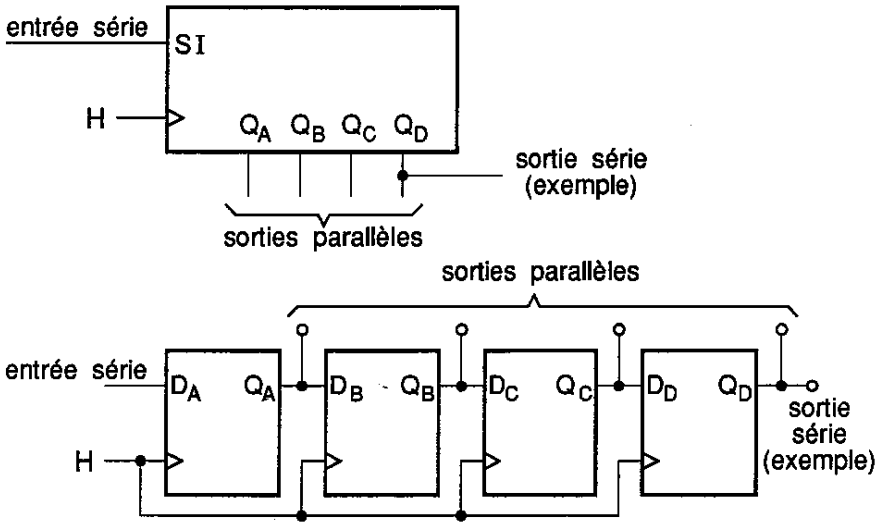
type d'opération
- série-série (SISO)
- série-parallèle (SIPO)
- parallèle-série (PISO)
- parallèle-parallèle (PIPO)

À cause du nombre limité de bornes ("pins") sur un boitier, il est difficile d'intégrer toutes les possibilités. D'ailleurs, des registres dits universels possèdent des modes d'opération offrant des avantages équivalents à un registre pouvant réaliser tous les types d'opération.

4.9.2 - Régistres série-série, série-parallèle

Les régistres série-série servent fréquemment de files d'attente ou de mémoires circulantes où la sortie est connectée à l'entrée pour permettre le blouclage de l'information. Les régistres série-parallèle peuvent servir à convertir une information sérielle sous forme parallèle à la réception d'un message sur une ligne de transmission.

Les régistres série-série et série-parallèle sont synthétisés de la même façon : une cascade de bascules "D" reliées en série; l'entrée série est l'entrée "D" de la première bascule ; la sortie série correspond à l'une des sorties des bascules (celle déphasée du nombre de période d'horloge désirée) ; les sorties parallèles sont formées par les sorties de toutes les bascules.

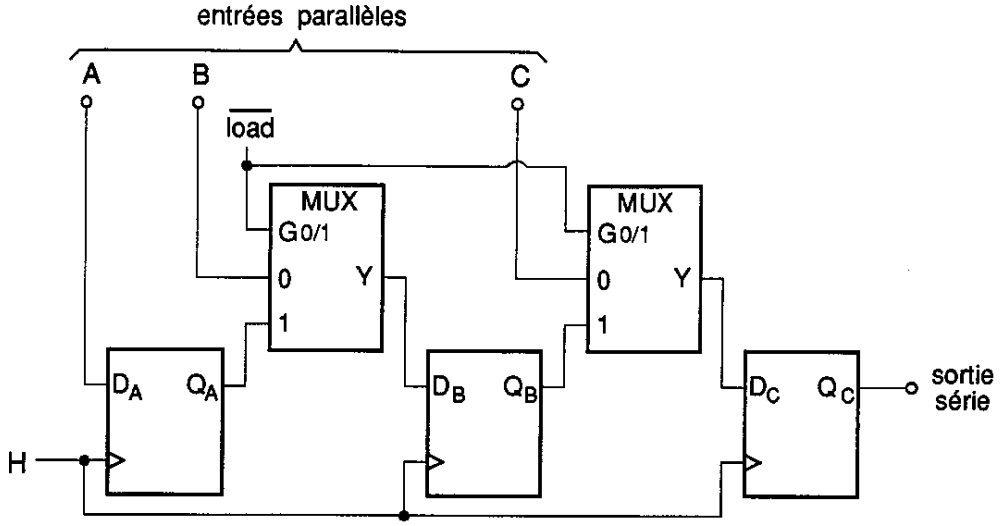


entrée série	état présent				état suivant			
	A	B	C	D	A	B	C	D
0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1
1	0	0	0	1	1	0	0	0
1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0
etc...								

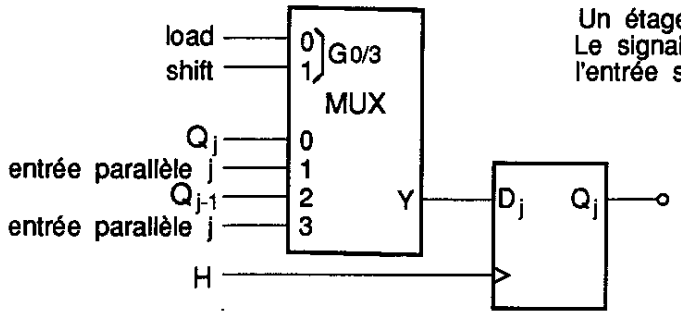
4.9.3 - Régistres parallèle-série, parallèle-parallèle

Ce type de régistre opère de la façon inverse à ceux du type série-parallèle: il peut prendre une information parallèle et la mettre en série en vue de la transmission sur une ligne. Sa conception est cependant plus compliquée. Encore ici, les régistres sont constitués d'une cascade de bascules "D" reliées en série pour pour pouvoir sortir l'information en série par décalage successif. Mais, pour enregistrer la donnée, il faut entrer chacun des bits à chacune des entrées "D" cor-

respondantes. Ainsi, il est nécessaire de sélectionner soit le bit de donnée, soit la sortie de la bascule précédente pour chaque étage. Cette sélection se réalise facilement par un multiplexeur 2 à 1 où l'entrée "adresse" est un signal de commande externe qui demande l'enregistrement de la donnée ("load").



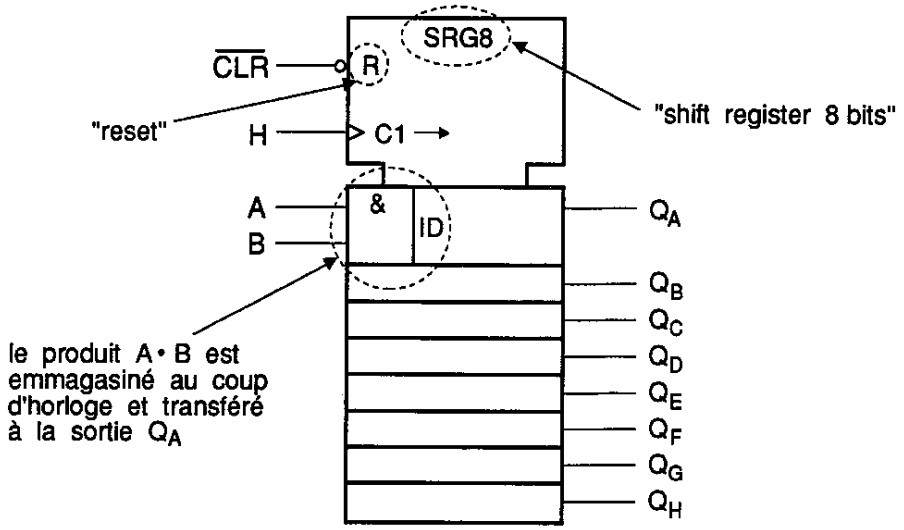
Les registres parallèle-parallèle simples se limitent à des bascules "D" placées en parallèle. Ils servent à enregistrer la donnée en parallèle pendant un certain laps de temps comme cela est si souvent nécessaire dans un système avec bus. Il est possible d'améliorer un registre parallèle-série pour le rendre parallèle-parallèle. Les sorties de chacune des bascules deviennent les sorties parallèles en plus d'être bouclées pour garder l'information telle qu'elle au coup d'horloge. Ce nouveau registre parallèle-parallèle avec entrée et sortie série, avec commande de chargement et de décalage porte le nom de registre universel. Il est la base des unités arithmétiques binaires.



Un étage de registre universel.
Le signal Q est remplacé par l'entrée série pour le 1er étage

4.9.4 - Exemples "TTL"

Le 74164 est un registre à décalage de 8 bits de type SISO-SIPO avec mise à zéro asynchrone. Au coup d'horloge, la sortie Q_A vaut $A \cdot B$ (A et B sont les entrées séries) et le décalage se fait de Q_A vers Q_H.

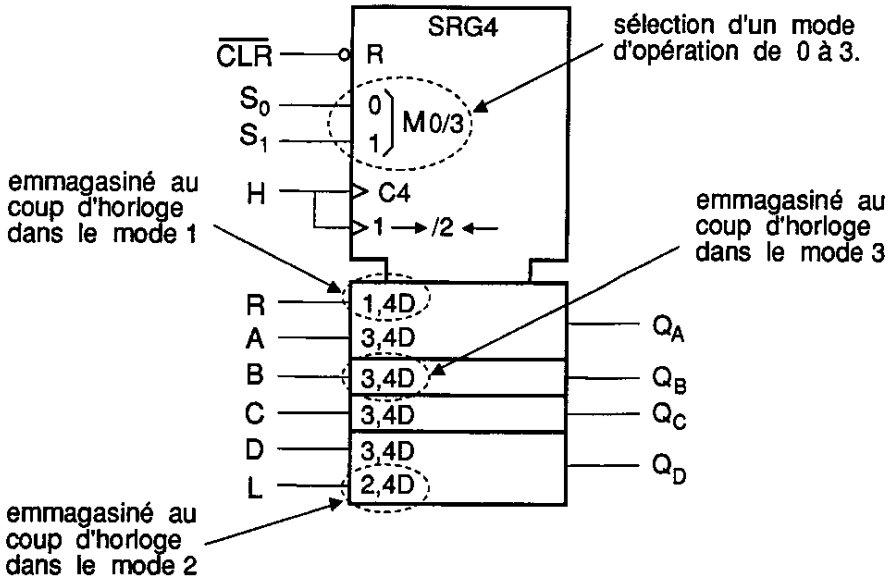


Le 74194 est un registre universel bidirectionnel à 4 bits. Il possède 4 entrées et 4 sorties parallèles ; 2 entrées séries R et L pour un décalage à droite ou à gauche respectivement (R ="right", L ="left") ; 2 commandes d'opération S_1 et S_0 qui ne doivent pas changer lorsque H = 0 (bascule J-K maître-esclave) et un signal de mise à zéro asynchrone.

Avec les 2 commandes d'opération, le 74194 agit selon 4 modes :

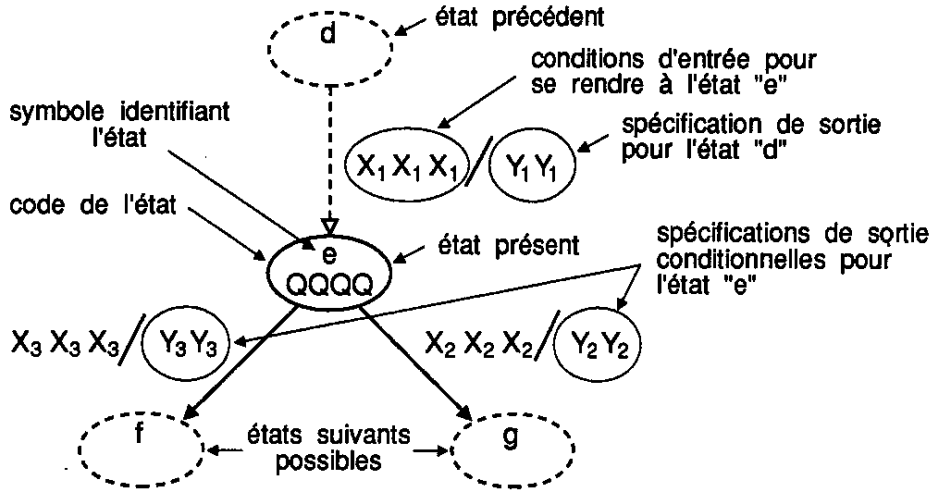
S_1	S_0	action
0	0	aucune
0	1	décalage à droite (Q_A vers Q_D)
1	0	décalage à gauche (Q_D vers Q_A)
1	1	chargement parallèle

	mode		entrées série		entrées parallèles				sorties			
$\overline{CLR}$	S_1	S_0	L	R	A	B	C	D	Q_A	Q_B	Q_C	Q_D
0	X	X	X	X	X	X	X	X	0	0	0	0
1	0	0	X	X	X	X	X	X	Q_{An}	Q_{Bn}	Q_{Cn}	Q_{Dn}
1	0	1	X	0	X	X	X	X	0	Q_{An}	Q_{Bn}	Q_{Cn}
1	0	1	X	1	X	X	X	X	1	Q_{An}	Q_{Bn}	Q_{Cn}
1	1	0	0	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	0
1	1	0	1	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	1
1	1	1	X	X	a	b	c	d	a	b	c	d



4.10 - Le diagramme d'état

Le diagramme d'état est d'une aide systématique dans l'analyse et la synthèse d'une machine séquentielle. Il est au circuit séquentiel ce que la table de vérité est au circuit combinatoire. En principe, le diagramme d'état possède aucune règle rigoureuse quant à sa représentation mais l'information qu'il contient doit indiquer clairement le cheminement des divers états de la machine. Le dessin reproduit plus bas montre une partie d'un diagramme d'état classique avec sa symbologie.



Dans ce diagramme, un état est représenté par une ellipse et les transitions entre les états sont indiquées par des lignes reliant les ellipses des états impliqués. Les chiffres binaires QQQQ sont le code de l'état (le code est souvent la valeur binaire des sorties des éléments "mémoire" qui forment le circuit séquentiel Quelquefois, ces sorties correspondent directement aux sorties de la machine mais ce n'est pas toujours le cas).

Il est à noter qu'il n'y a pas plus d'information dans un diagramme d'état que dans une table d'état, cette dernière indiquant l'état suivant dépendamment de l'état présent et des commandes d'entrée. Seule la représentation change.

Exemple #1 :

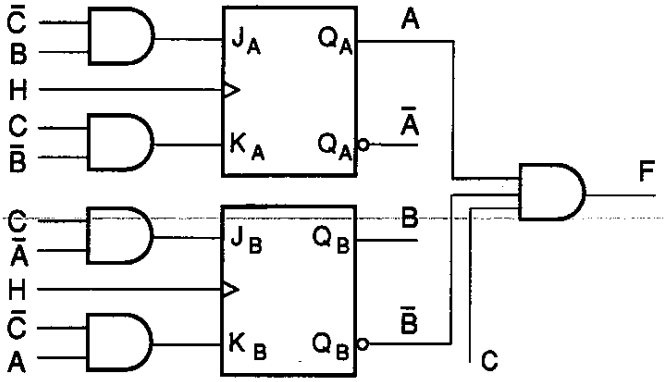
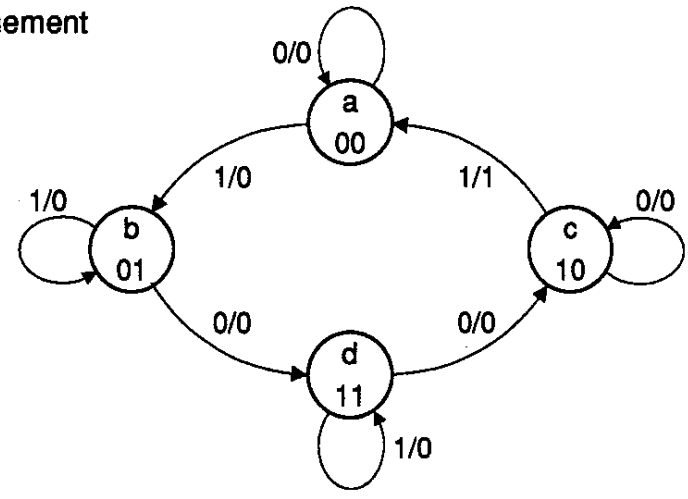


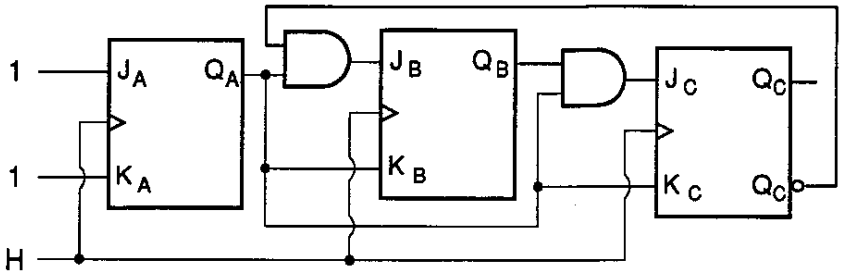
table d'état état présent A	B	C	J_A	K_A	J_B	K_B	état suivant A	B	F
a	0	0	0	0	0	0	0	0	0
	0	0	1	0	1	1	0	1	0
b	0	1	0	1	0	0	1	1	0
	0	1	1	0	0	1	0	1	0
c	1	0	0	0	0	0	1	0	0
	1	0	1	0	1	0	0	0	1
d	1	1	0	1	0	0	1	0	0
	1	1	1	0	0	0	1	1	0

diagramme d'état

format de branchement
C/F

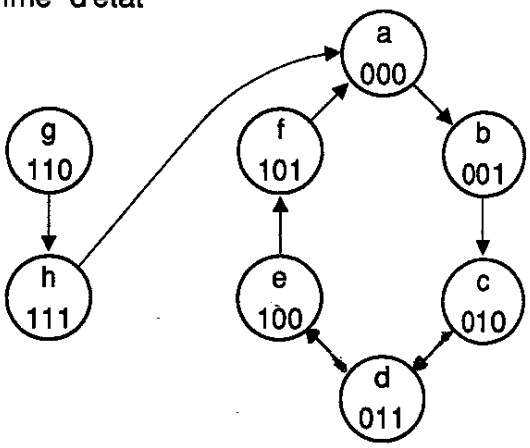


Exemple #2 : Dans un compteur, les sorties sont les bits de codage de l'état (on dit qu'il n'y a pas de décodeur de sortie). Ces bits correspondent à la sortie des bascules ou éléments "mémoire".



code de l'état = $Q_C Q_B Q_A$

diagramme d'état



Le diagramme d'état n'a pas de format de branchement : il est réduit à sa plus simple expression. À chaque coup d'horloge, le circuit change d'état car il n'y a pas de conditions d'entrée pour se rendre à un état. De plus, comme les sorties sont les bits de codage, il n'y a pas lieu d'indiquer les spécifications de sortie.

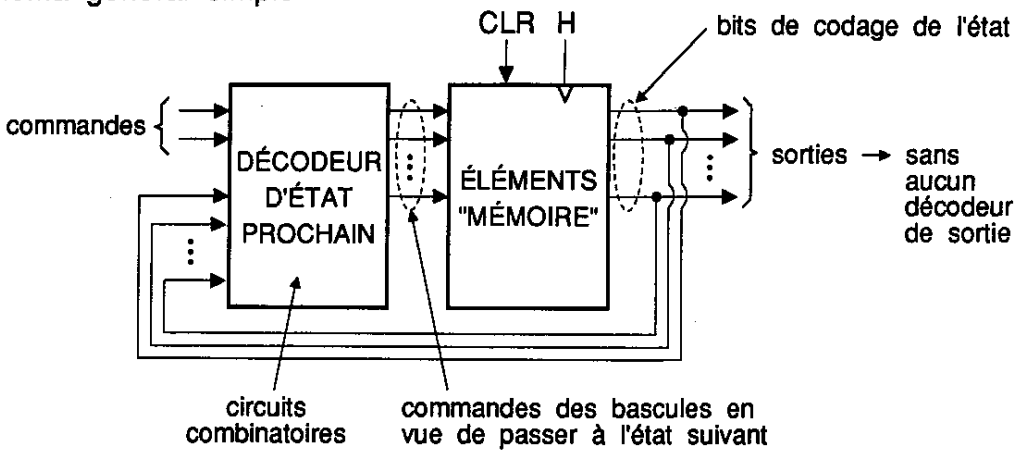
Vérifiez ce diagramme d'état.

4.11 - Les compteurs

4.11.1 - Définition

Un compteur est un circuit séquentiel synchrone qui assure la fonction de comptage (binaire ou autre). Il se compose de n éléments "mémoire" ou "n" représente le nombre de bits du compteur. Il est donc possible d'obtenir un maximum de 2^n états. Le compteur a la particularité de n'avoir aucun décodeur de sortie (cf. chapitre sur les circuits séquentiels synchrones). La seule sortie qui nous intéresse étant l'état des bascules à chaque période de l'horloge.

Schéma général simple

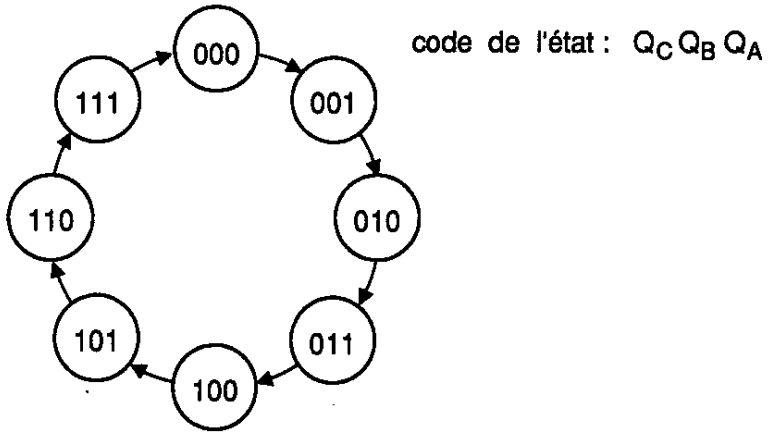


L'exemple #2 du paragraphe 4.10 est un compteur binaire (le compteur compte en incrémentant la valeur binaire des bits de codage de l'état) à 3 bits modulo 6 (6 états, les 2 autres pouvant se produire au départ seulement). Remarquez que l'horloge agit directement sur les éléments "mémoire".

→ Il ne faut pas toucher à l'horloge ←

Ce qui signifie qu'il ne faut pas, en autre, utiliser le signal d'horloge comme signal d'entrée de portes pour calculer des états suivants à cause des propriétés dynamiques des portes (i.e. délais de propagation). Nous verrons d'autres exemples lorsque nous couvrirons le sujet des machines séquentielles algorithmiques et les horloges multiphases.

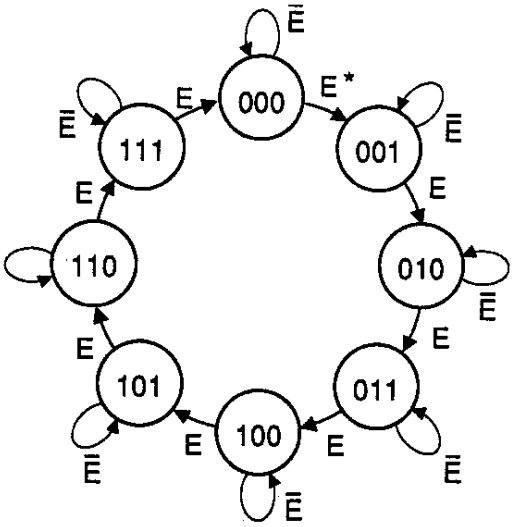
Exemple : compteur binaire 3 bits complet (8 états) avec J-K



Si on veut bloquer le compteur par une commande synchrone "E" telle que :

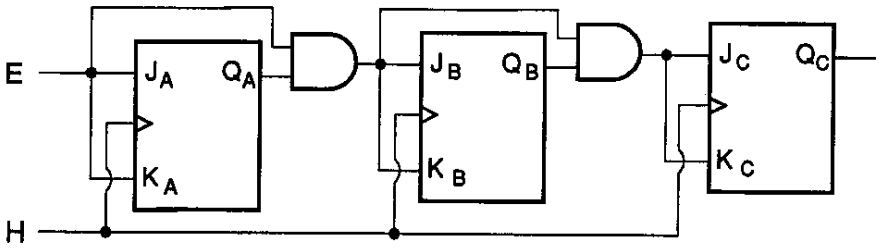
E = 1 → compte normalement
E = 0 → arrêt, garde le compte précédent

on obtient le diagramme d'état suivant :



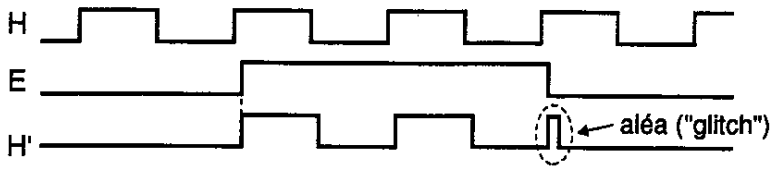
* le format de branchement montre que la transition se fait dépendamment de la commande "E". Les spécifications de sortie ne sont pas indiquées: il s'agit d'un compteur.

Le truc pour ajouter l'entrée "E" est de forcer à "0" toutes les commandes J et K des bascules ce qui oblige les bascules à ne faire aucune action lorque "E = 0".



Il ne fallait pas arrêter le comptage en bloquant l'horloge de cette manière :

puis en se servant de H' comme nouvelle horloge du système car puisque E est en retard sur H (E est synchronisé par H) ce qui produit un aléa sur H' lorsque E retourne à "0" sur la montée de H.

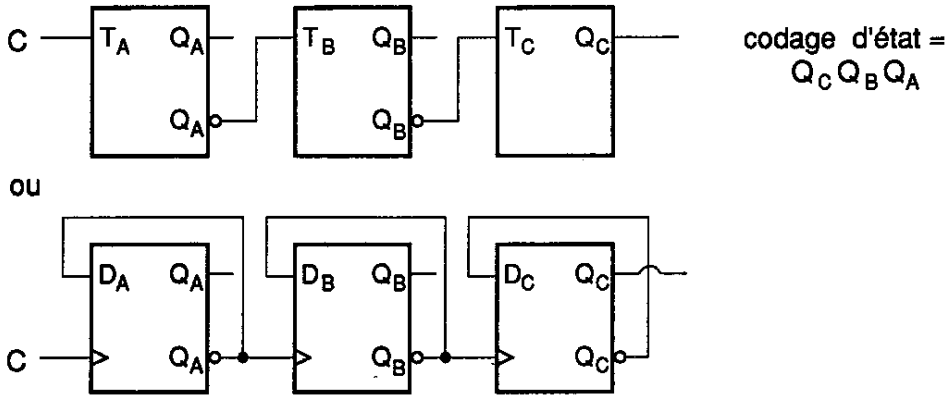


Les aléas sur H' se multiplient lorsqu'on admet que le signal E peut lui-même avoir des aléas n'a pas d'importance dans un ciruit synchrone en autant qu'ils aient disparu au prochain coup d'horloge. Mais le signal d'horloge doit être dépourvu d'aléa.

4.11.2 - Compteurs série (non synchrone)

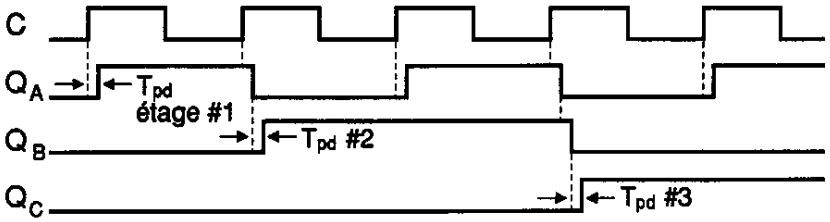
Les compteurs série sont composés de bascules T ("toggle") connectées bout à bout, chacune étant synchronisée par la sortie de la précédente. Ainsi, chaque étage forme un compteur modulo 2 avec horloge 2 fois plus lente d'étage en étage. Les compteurs sériels ne sont donc pas synchrones.

Exemple : compteur série 3 bits complet



ou

diagramme de fonctionnement



Ce type de compteur, n'a très peu d'adeptes :

- il produit une séquence binaire forcée
- il atteint une fréquence maximale faible

$$f_{max} = \frac{1}{\sum_{i=1}^{n} T_{pd_i}}$$ (T_{pd_i} = temps de propagation d'un étage)

Exemple : compteur série de 8 étages (n=8)

T_{pd} ~ 25 nS

alors f_{max} ~ 5 MHz

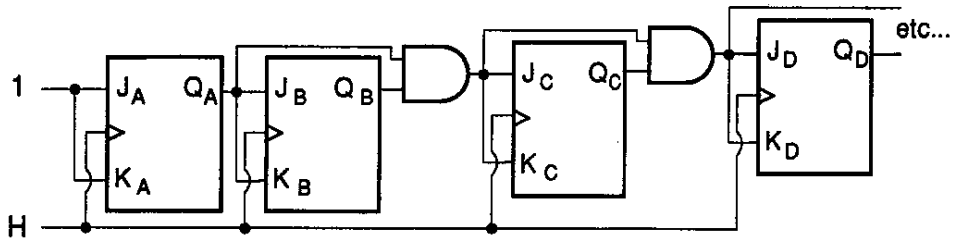
- plus le nombre d'étages ↑, moins f_{max} est grand donc $f_{max} \alpha \frac{1}{n}$ 

- décodage combinatoire des états difficile sans aléa
- manque de synchronisation avec la commande "C"
- troncature sans aléa (état transitoire) difficile

4.11.3 - Compteurs binaires

Un compteur binaire compte de telle façon que la valeur binaire des bits de codage croit ou décroit d'un unité à la fois. Le compteur peut être tronqué i.e. le comptage saute volontairement une partie de sa séquence. La longueur de la séquence est le module.

Pour un compteur à module maximum (complet) qui compte croissant (décroissant), la sortie d'un étage renverse lorsque les sorties de tous les étages précédents sont à "1" ("0"). Ceci nous suggère une idée pour la synthèse d'un tel compteur, elle apparait ici bas :

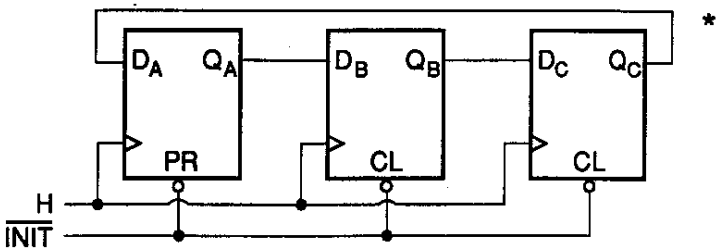


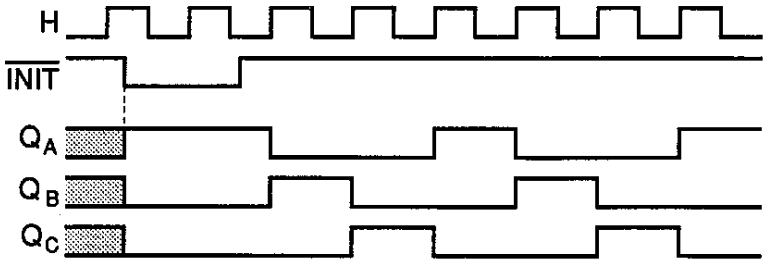
Tronquer un compteur binaire peut s'effectuer en ajoutant un circuit de détection de l'état de sortie. Ce dernier envoie un signal appliqué sur les mises à zéro de toutes les bascules pour remettre le compteur à zéro. Cette technique est à éviter sur un circuit synchrone : le compteur passe transitoirement par l'état suivant l'état de sortie. Il faut utiliser une autre méthode qui sera discutée plus tard.

4.11.4 - Compteurs en anneau ("ring counter")

Un compteur en anneau est un compteur qui fait "circuler" un "1" (ou un "0"). C'est un compteur qui possède autant d'états qu'il y a d'étages. Cette caractéristique semble un désavantage mais elle permet l'élimination du décodage d'état lorsqu'on emploie le compteur dans un séquenceur (cf. chapître 8 sur les séquenceurs).

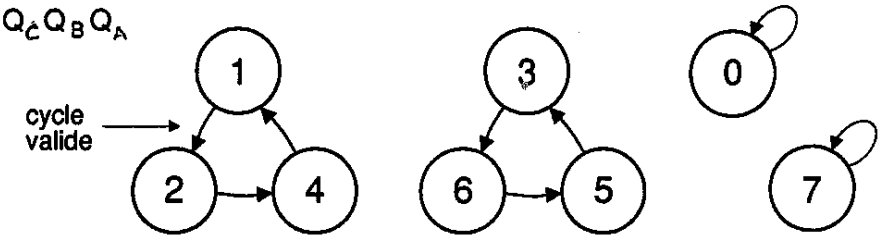
La figure ci-dessous montre un compteur en anneau de 3 bits avec son diagramme de fonctionnement. Notez que le compteur procure plusieurs cycles fermés mais on ne considère qu'un seul est valide. On doit donc faire une initialisation se servant des entrées asynchrones des bascules ou faire une autocorrection.





* le compteur fait circuler un "1"
  initialisation avec les entrées asynchrones

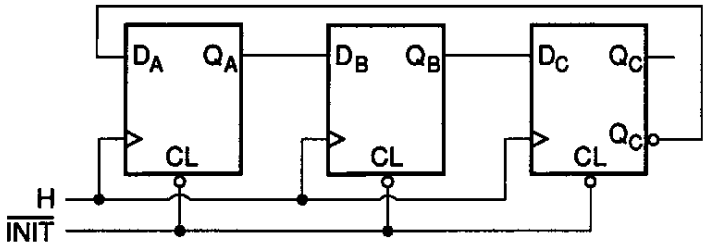
codage $Q_C Q_B Q_A$



4.11.5 - Compteurs Johnson

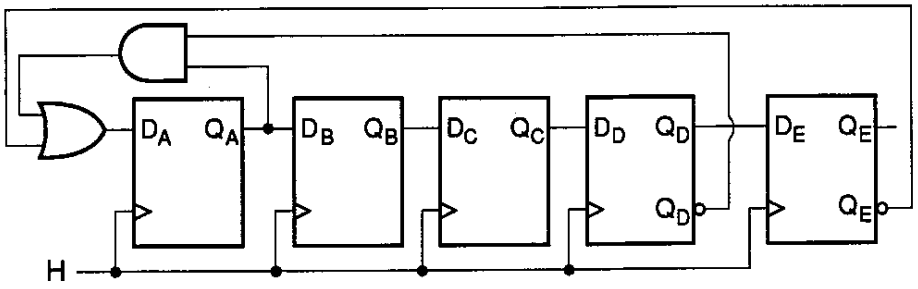
Les compteurs Johnson ressemblent aux compteurs en anneau sauf qu'ils offrent 2 fois plus d'états pour le même nombre d'étages. Le décodage des états demeure relativement simple, limité à une porte par état. Les compteurs Johnson à 5 étages sont fréquemment utilisés comme compteurs décimaux car ils possèdent 10 états convertis en séquence binaire de 0 à 9 avec l'aide d'un décodeur X / Y.

Encore avec les compteurs Johnson, plusieurs cycles distincts. Donc, si par hasard le compteur entre dans un cycle invalide au départ, il y restera tant qu'on n'interviendra pas pour l'en ressortir. Les 2 méthodes proposées pour les compteurs en anneau conviennent.



cycle Q_A	Q_B	Q_C	décodeur d'état
0	0	0	$\overline{Q}_A\overline{Q}_C$
1	0	0	$Q_A\overline{Q}_B$
1	1	0	$Q_B\overline{Q}_C$
1	1	1	Q_AQ_C
0	1	1	$\overline{Q}_AQ_B$
0	0	1	$\overline{Q}_BQ_C$

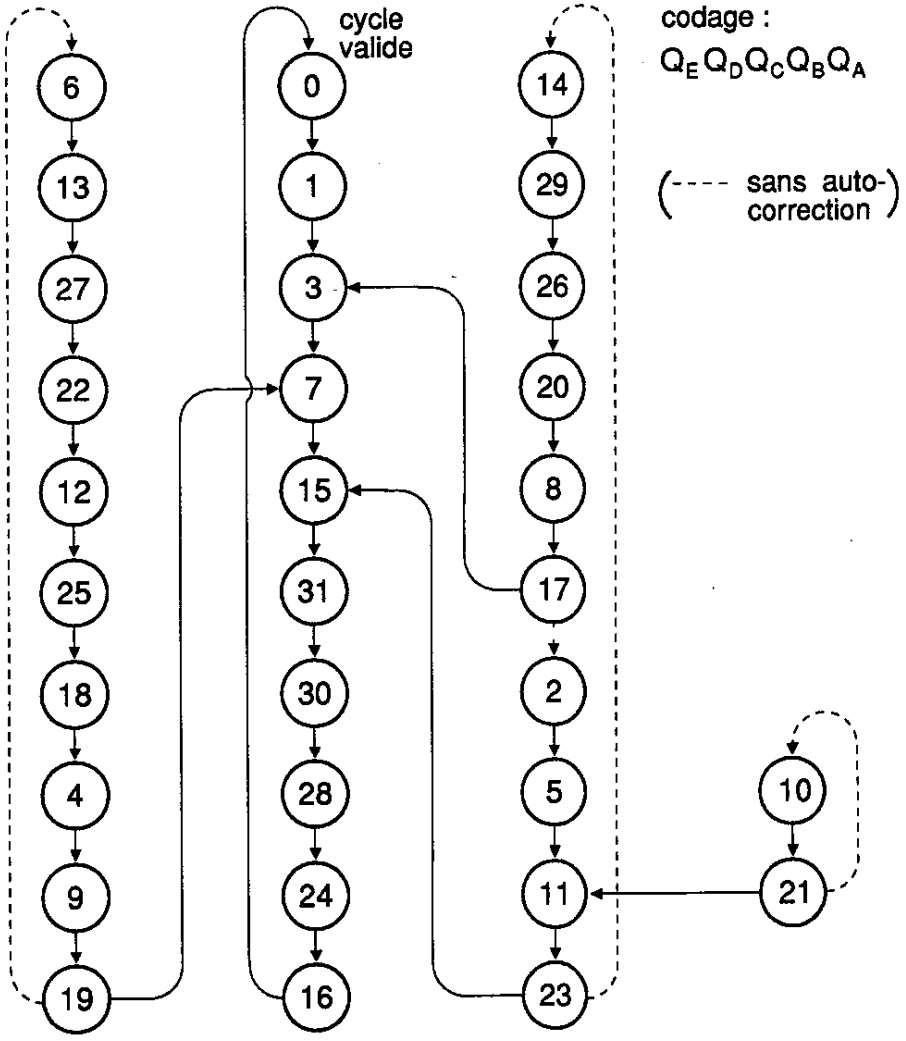
Exemple : Compteur "Johnson" 5 bits avec autocorrection



au lieu d'obtenir l'expression $D_A = \bar{Q}_E$

on a : $D_A = \bar{Q}_E + \bar{Q}_D \cdot Q_A$

le terme $\bar{Q}_D \cdot Q_A$, assure la sortie des états indésirables.



4.11.6 - Exemples TTL

Le 74163 est un compteur binaire de 4 bits entièrement synchrone. Toutes les commandes externes ("load", "clear" ...) sont exécutées uniquement sur la montée de l'horloge. Il n'y a donc pas d'entrées asynchrones.

Ce compteur est programmable en ce sens qu'il peut être placé dans n'importe lequel des 16 états par une demande de chargement et en plaçant sur les 4 entrées parallèles le code de l'état. L'action sera effectuée au coup d'horloge, d'un autre côté, le compteur possède une commande de mise à zéro synchrone de sorte qu'on peut s'en servir dans un circuit et demeurer synchrone. Finalement, le 74163 possède aussi 2 commandes d'inhibition (ENP, ENT) qui bloquent le compteur. La différence entre ces 2 commandes d'inhibition se situe au niveau de l'affectation de "ENT" face à la retenue. Il faut noter que la mise à zéro est prioritaire sur le chargement qui, à son tour, est prioritaire sur l'inhibition.

Opérations :

$\overline{CL}$	$\overline{LD}$	ENP·T	ACTION
0	X	X	mise à zéro
1	0	X	chargement
1	1	1	compte

CO ("carry out" ou retenue) = $Q_A Q_B Q_C Q_D \cdot$ (ENT)
= décodeur de l'état 15
il dure une période
d'horloge ("blip")

Il est possible de relier plusieurs 74163 en cascade de manière à réaliser un compteur de 8, 12, 16 ... bits. Le tout est rendu possible grâce à la sortie de retenue et les commandes d'inhibition.

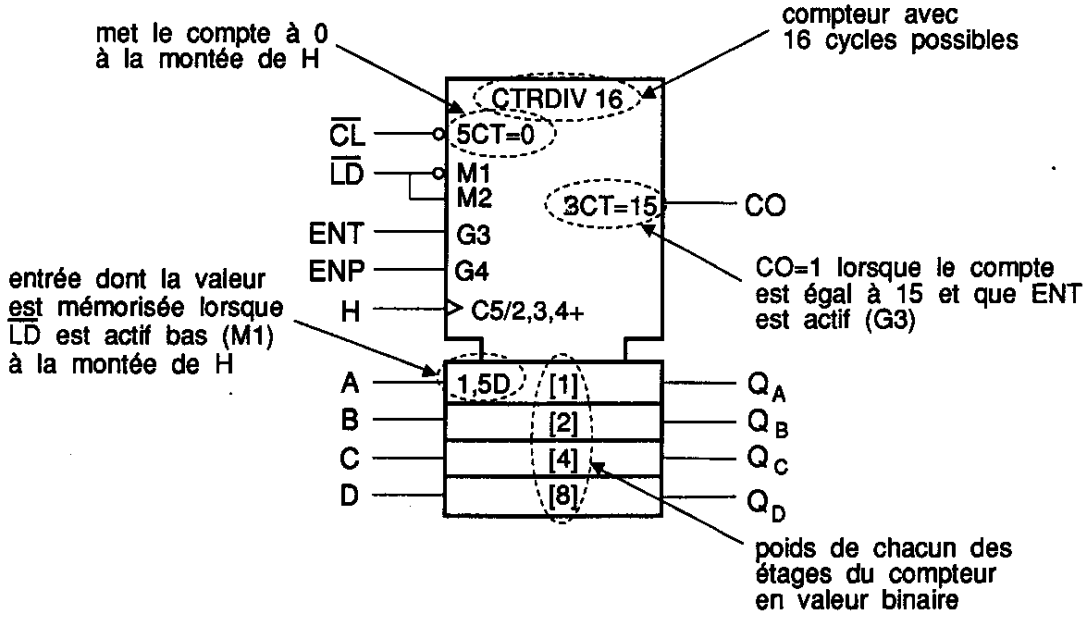
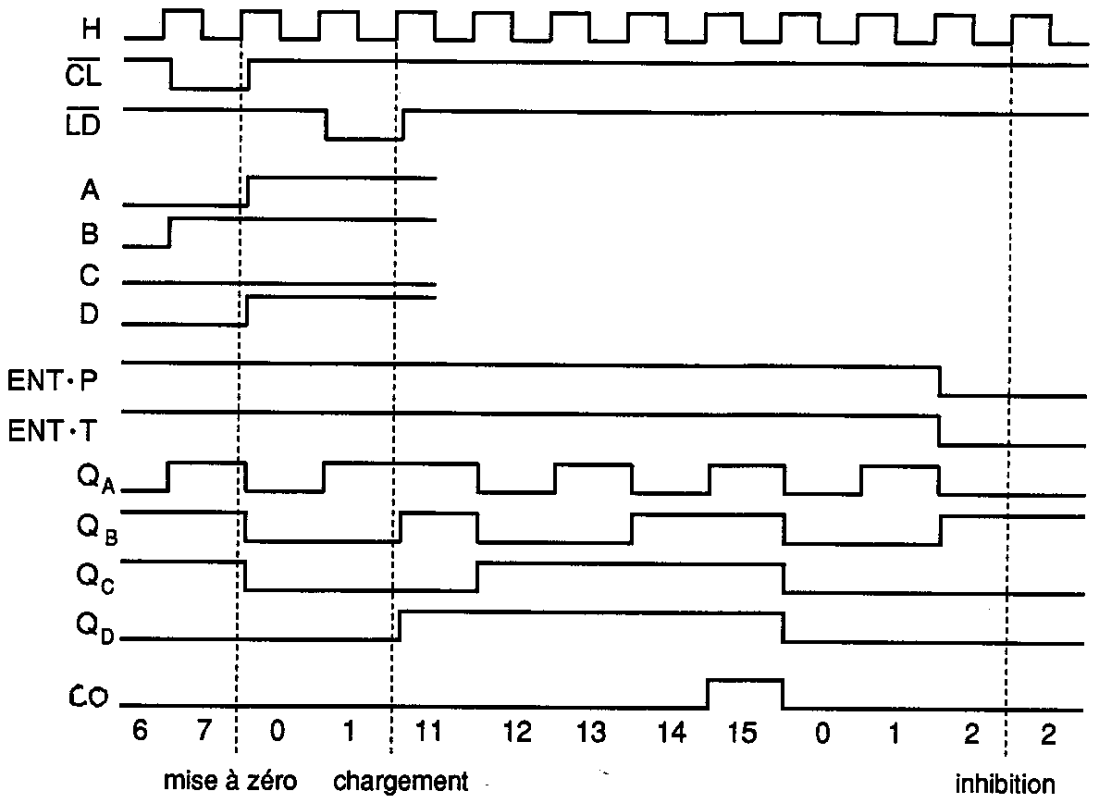


diagramme de fonctionnement



Le 74169 est un compteur 4 bits ressemblant au 74163 au point de vue fonctionnement, à la différence près qu'il peut compter autant en ordre croissant qu'en ordre décroissant. C'est donc un compteur 4 bits "UP/DOWN" entièrement synchrone, avec retenue lorsque le compte est égal à 15 (0) en comptant en croissant (décroissant) ce qui permet la cascade à 8, 12, 16... bits avec les commandes d'inhibition "ENT et ENP".

Exemple #1 : Compteur binaire modulo 6 (0→5)

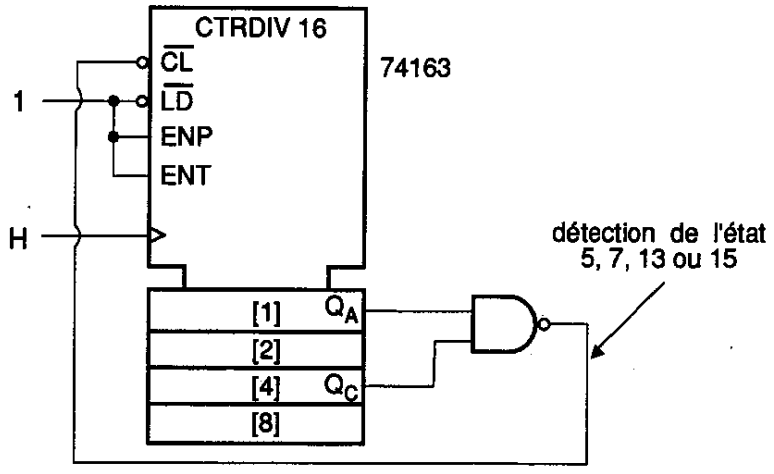
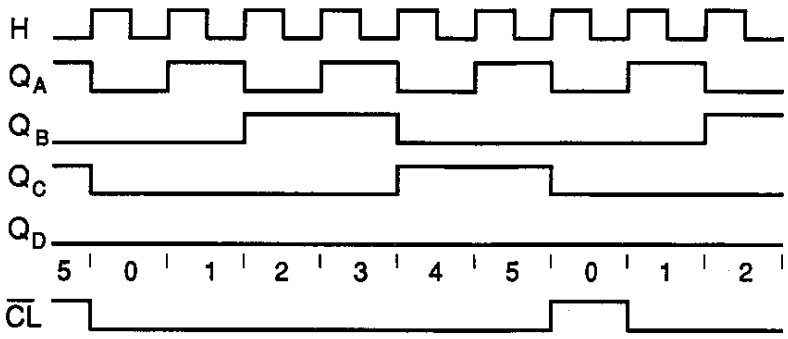
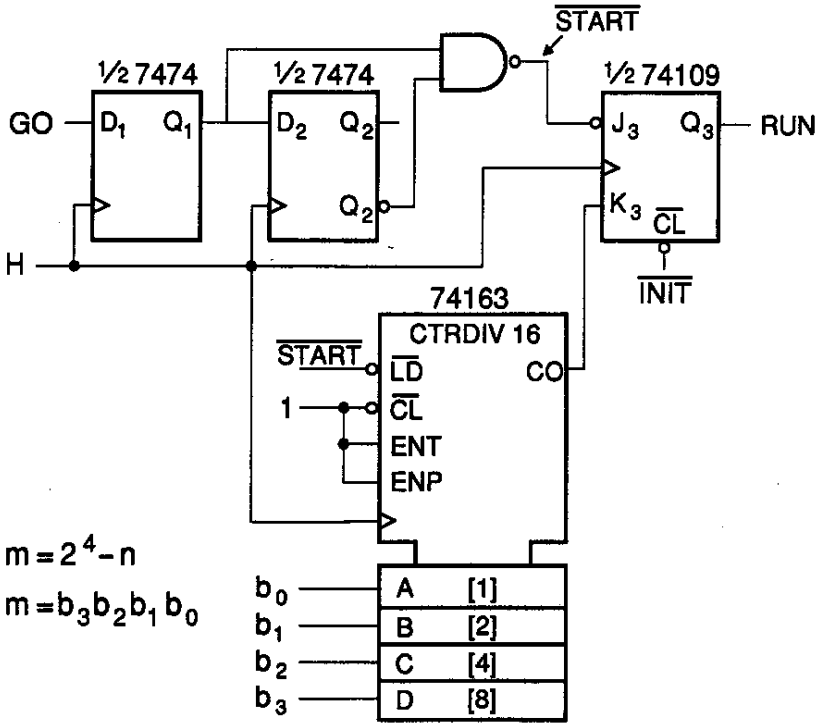


diagramme de fonctionnement



Exemple #2 : On veut produire un signal synchrone qui dure "n" coups d'horloge ($1 \leq n \leq 16$).
Le nombre "n" est introduit sur 4 interrupteurs.
Un interrupteur fait l'initialisation tandis qu'un autre demande le départ.



Le nombre "m" est déduit de "n" par $m=2^4-n$ sur 4 bits
Le signal "GO" provient d'un interrupteur. Il est donc asynchrone d'où l'emploi d'une première bascule "D" pour le synchroniser. La deuxième bascule "D" produit un bip (différienciateur de front montant) "$\overline{START}$". Ce dernier active le signal "RUN" et demande le chargement de "m". Après "n" coups d'horloge, le compteur atteint le compte de 15, émet une retenue qui désactive le signal "RUN".

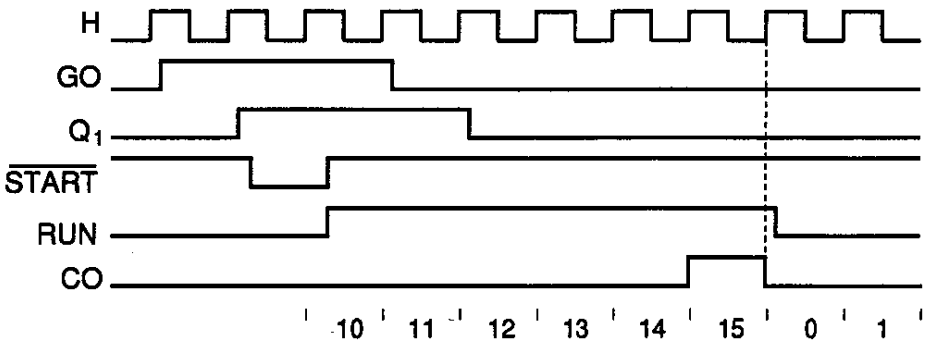
Note : Transfert J - $\bar{K}$ à $\bar{J}$ - K

J	$\bar{K}$	action sur Q
0	0	mise à zéro
0	1	aucune
1	0	renverse
1	1	mise à un

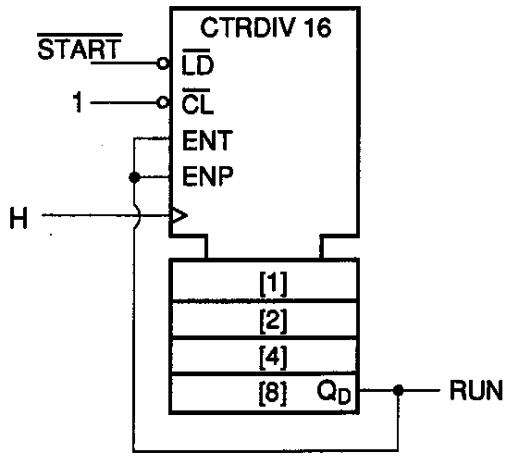
$\bar{J}(\bar{K})$	K(J)	action sur $\bar{Q}$
0	0	mise à un
0	1	renverse
1	0	aucune
1	1	mise à zéro

la table est correcte en prenant $\bar{Q}$ du 74109 pour Q lorsqu'on utilise la bascule selon le mode J-K.

avec n = 6 m = 1010 A=0; B=1; C=0; D=1.



Pour n>16, on doit utiliser une cascade de 74163 ce qui permet de travailler sur plus de 4 bits. Par contre, avec 1≤ n≤8, on peut éliminer la bascule J-K et la remplacer en prenant la sortie Q_D pour le signal "RUN" et en reliant Q_D avec les bits d'inhibition.



Département de Génie électrique

GEL-10279 CIRCUITS LOGIQUES

NOTES DE COURS

CHAPITRE 5

ROM, RAM, PLA

5.1 Circuits à large intégration

Mémoire
- RAM (random access memory)
  - mémoire vive
  - lecture/écriture avec accès aléatoire
  - dynamique (DRAM)
    - grande capacité (512K)
    - demande un rafraîchissement exploitation plus complexe
  - pseudo-statique (VSRAM)
    - circuit de rafraîchissement inclus
  - statique (SRAM)
    - capacité plus faible (64K)
    - bascules "latch D"
- ROM (read only memory)
  - mémoire morte
  - lecture avec accès aléatoire
  - ROM
  - PROM
    - "programmable read only memory"
  - EPROM
    - "erasable programmable read only memory" (UV)
  - EEPROM
    - "electrically EPROM"

Synthèse de circuit combinatoire
- PLA (programmable logic array)
- PAL (programmable array logic) moins de possibilités que les PLA mais moins coûteux.

Ces circuits intégrés existent dans plusieurs des familles technologiques [TTL, ECL (moins), CMOS, NMOS, HMOS (surtout)].

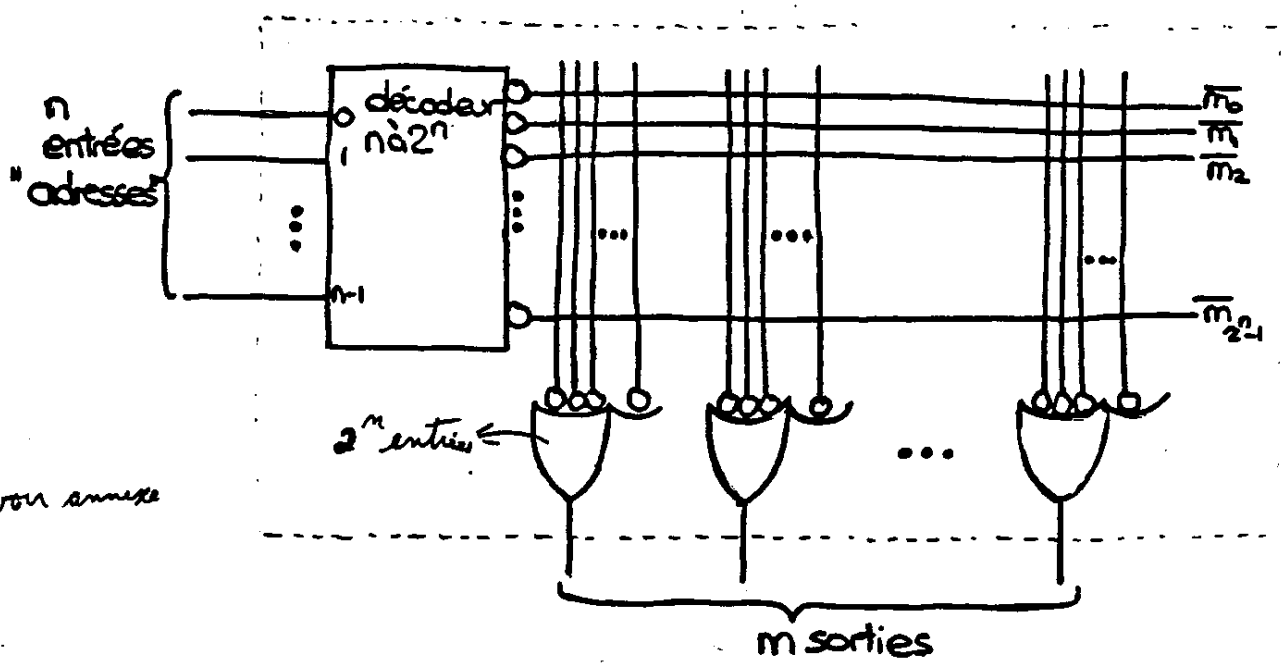
5.1.1 ROM

Définition

- dictionnaire numérique; table de vérité à n V.I. et m sorties
- circuit dont le code de la sortie dépend du code fourni à l'entrée;
- mémoire morte: elle contient une information permanente.

- Les ROM's sont des circuits combinatoires à 2 couches. La première est un décodeur à n adresses qui produit 2^n monômes (les minterms). La deuxième est programmable et elle fait la somme des monômes désirés pour générer la fonction.

Principe de base



non annexe

ROM $2^n \times m$

capacité

Exemples :

TTL 74188	PROM 32 x 8 (256 bits)
HMOS 8755	EPROM 2K x 8 (16,384 bits) Intel
TMS 2516	EPROM 2K x 8
TMS 2532	EPROM 4K x 8
TMS 2564	EPROM 8K x 8
TMS 27256	EPROM 32K x 8

etc.

Programmation

Les sorties du décodeur ne sont reliées, initialement, à aucune des 2^n entrées de chaque porte AND. Il faut faire les interconnexions nécessaires.

ROM : les interconnexions sont faites à l'usine par un procédé dit "par masque"

PROM : les interconnexions y sont toutes au départ. Chacune d'elles est un fusible que l'on fait sauter lorsqu'on ne veut pas de cette interconnexion. Le procédé peut se réaliser sur un appareil "portatif": un brûleur de PROM.

EPROM: chaque interconnection est réalisée à partir d'un transistor à effet de champ ayant une "gate" spéciale commandée par une charge électrique. Cette "gate" permet ou non l'interconnexion et cette dernière peut être effacée par la lumière ultra-violette (20 min. d'exposition).

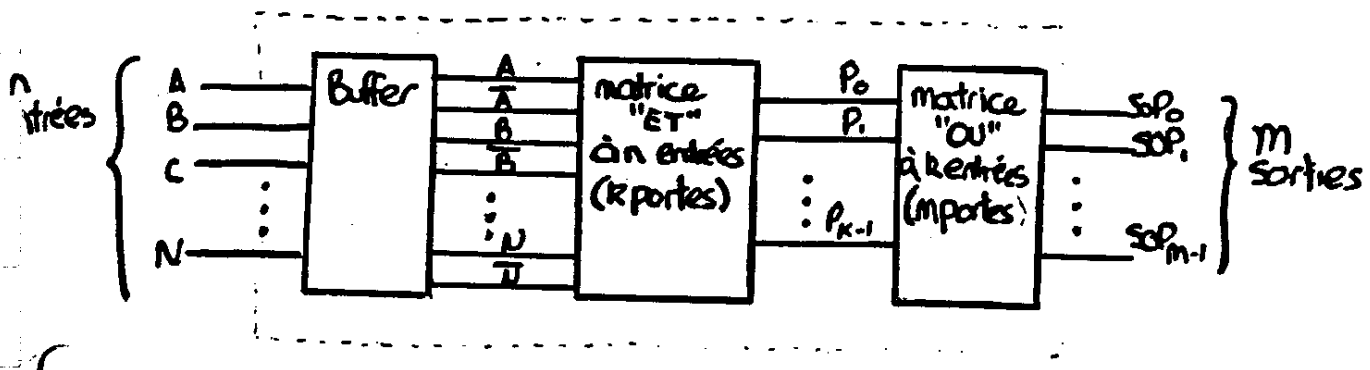
5.1.2 PLA

Définition

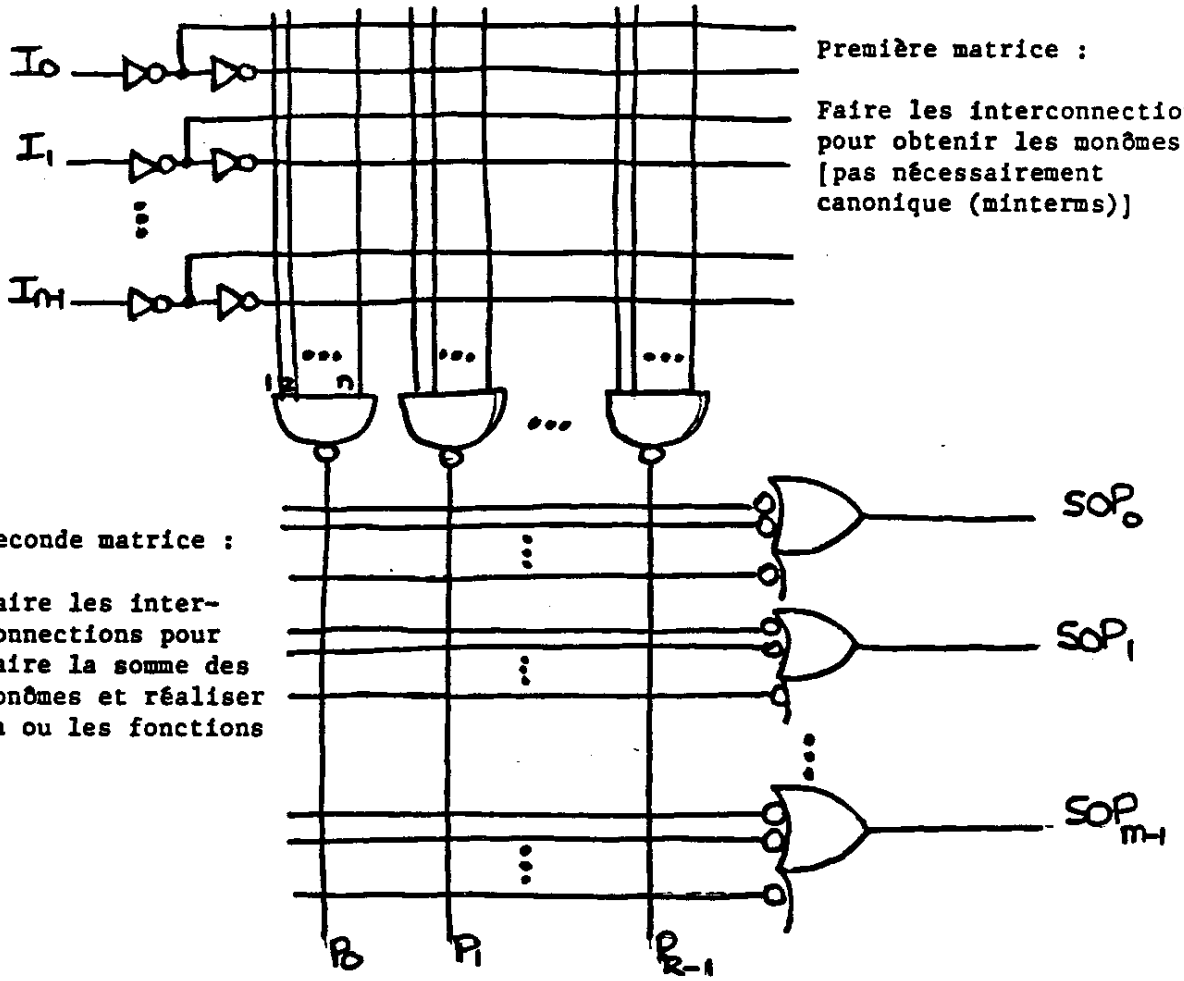
- circuit combinatoire le plus sophistiqué dans les LSI;
- conçu pour l'implantation d'expressions logiques du type somme de produits (SOP) de la même manière qu'un ROM mais il ne procure pas tous les minterms. De toute façon, la génération de tous les monômes n'est pas nécessaire dans bien des cas.
- Les PLA possèdent 2 champs programmables
  a) on peut déterminer les produits générés → matrice "ET"
  b) on peut choisir les entrées des portes "OU" → matrice "OU"

Différences avec ROM
- PLA peut avoir plus d'entrées adresses qu'un ROM (> 16) 2^{16} sorties du décodeur dans ROM
  2^{16} entrées de chaque "OU" dans ROM } 65 536 minterms à générer
  ce qui dépasse les capacités d'un ROM.
- Le PLA exige la simplification des expressions logiques avant l'implantation pour minimiser le nombre de monômes utilisés. Le PLA est plus efficace qu'un ROM pour la synthèse d'un circuit combinatoire.

Réalisation de principe (version simple)



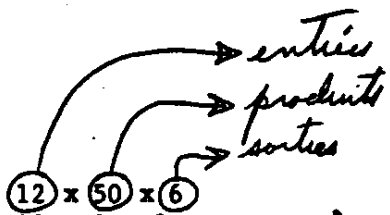
Principe de base (version simple)



PLA n x k x m

capacité

Exemples



TTL 74330 (12) x (50) x (6)

PAL 10H8 10 x 2 x 8

PAL 12H6 12 x (2 ou 3) x 6

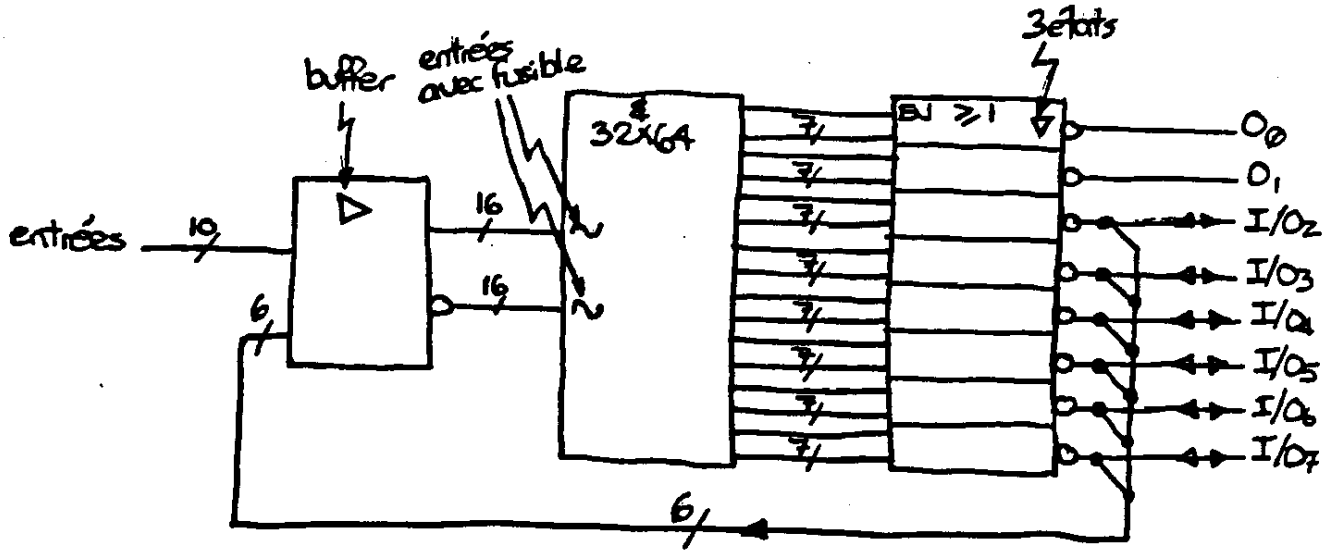
PAL 14H4 14 x 4 x 4

} Monolithic memories

etc.

Exemple

TIB PAL 16L8-20

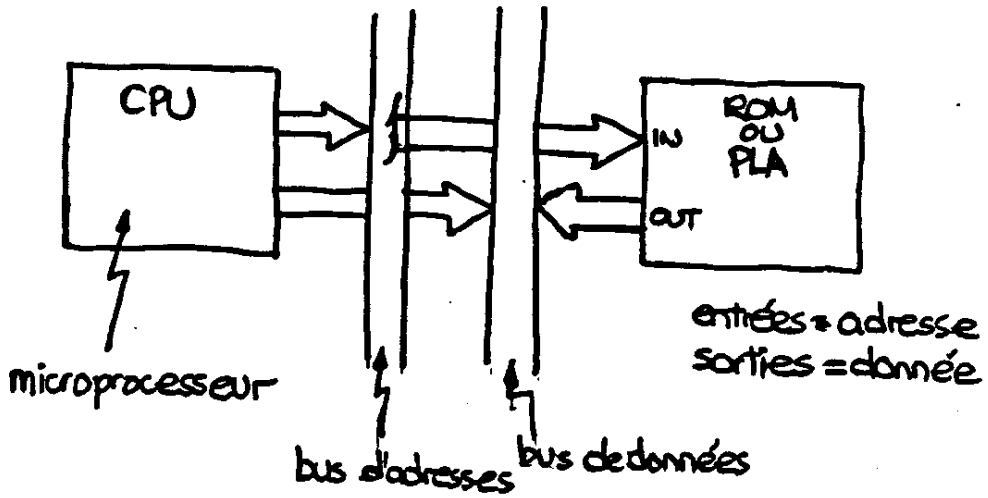


Utilisations

Même que ROM
- synthèse circuits combinatoires
- conversion de code (codage)
- table
- microprogrammation

\+ protection matérielle pour progiciel (de plus en plus).

Microprogrammation :

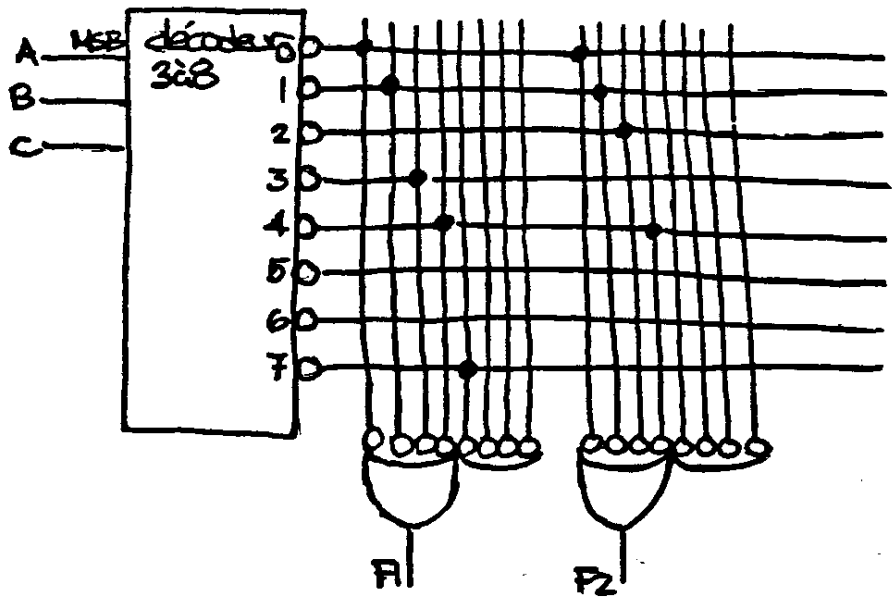


Circuits combinatoires

F1 = F(A,B,C) = $\Sigma_m(0,1,3,4,7)$

F2 = F(A,B,C) = $\Sigma_m(0,1,2,4)$

ROM



Capacité: au moins (8) x (2)

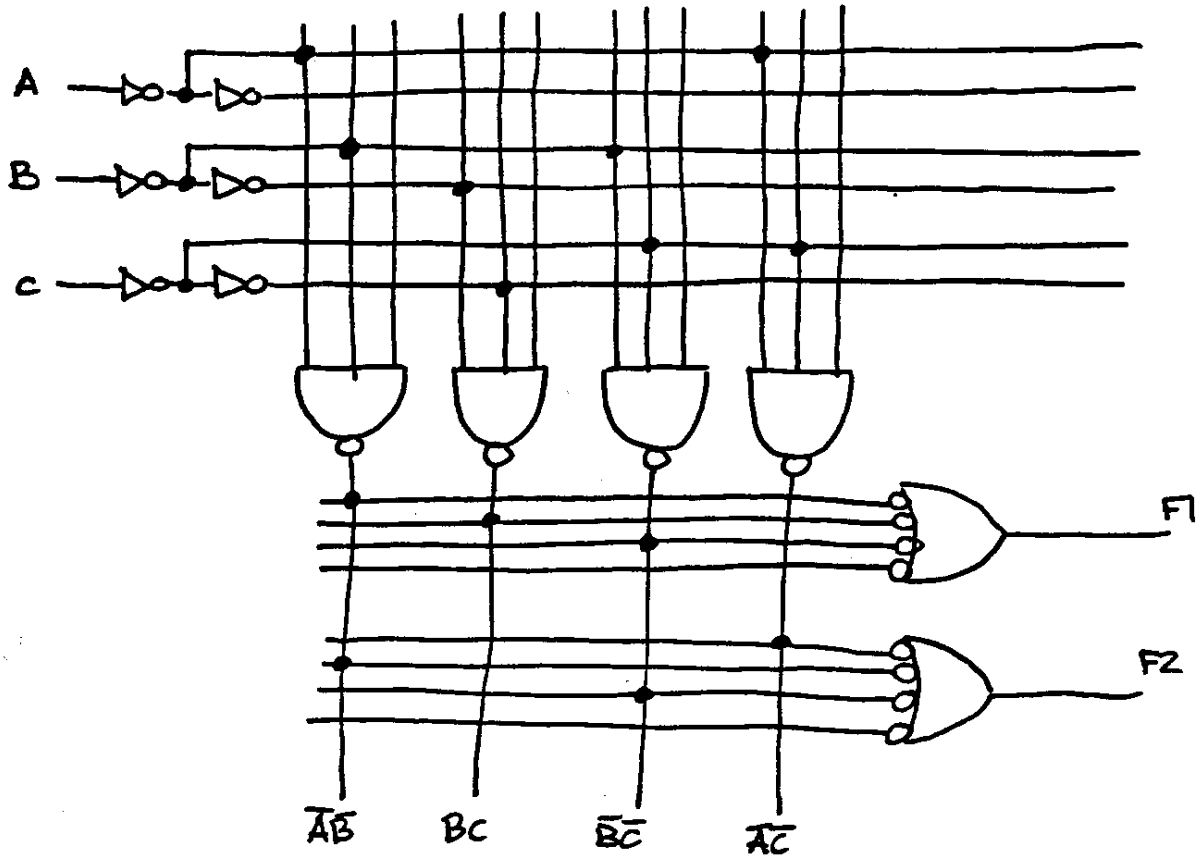
(8) → mots

(2) → sorties

PLA

F1 = $\overline{A}\overline{B} + BC + B\overline{C}$ (Karnaugh)

F2 = $\overline{A}\overline{C} + \overline{A}B + B\overline{C}$ (Karnaugh)



Capacité: au moins 3 x 4 x 2

5.1.3 PAL

- Approche similaire au PLA sauf que la région où est définie la matrice de OU n'est pas programmable.
- Un peu moins flexible que le PLA mais moins difficile à programmer et moins couteux. A ce titre, il s'apparente au PROM.
- Les sorties peuvent maintenant être synchronisées de façon à réaliser des comptems ou autres séquencems.

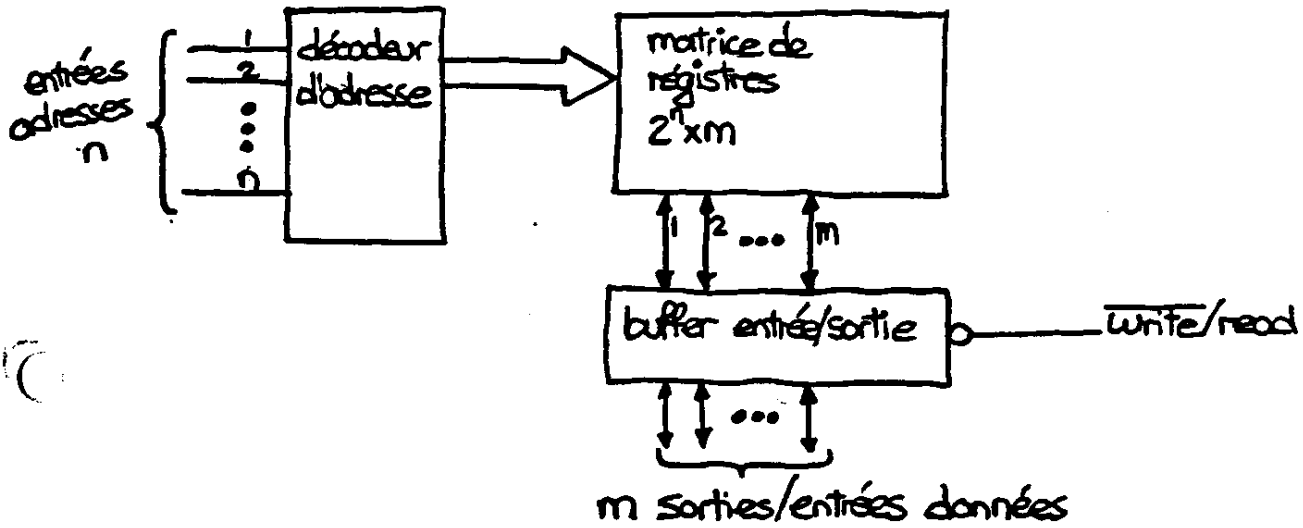
schéma voir annexe

5.1.4 RAM

Définition

- circuit LSI à mémoire, permettant l'écriture et la lecture d'une information;
- le contenu disparaît lorsque l'alimentation manque;
- deux types → RAM dynamique (DRAM) - grande capacité (256K bits) (même 512K, 1024K)
  - plus difficile d'exploitation car elle demande un rafraîchissement
  - RAM statique (SRAM) - capacité moindre (64K)
  - réalisée à partir de bascules "latch D".
- les entrées doivent être démultiplexées et les sorties, multiplexées.

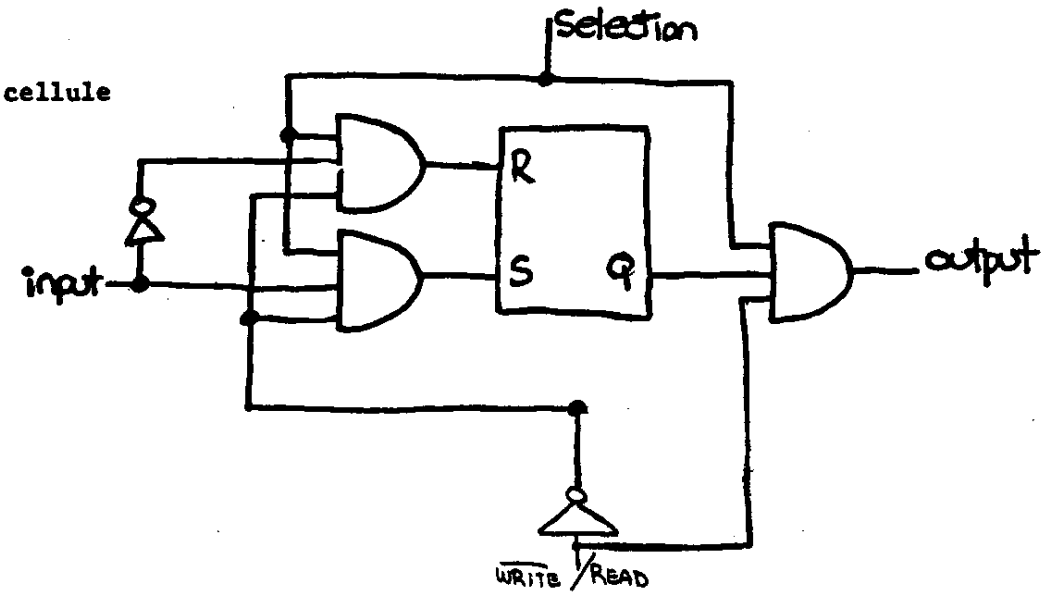
Blocs fonctionnels

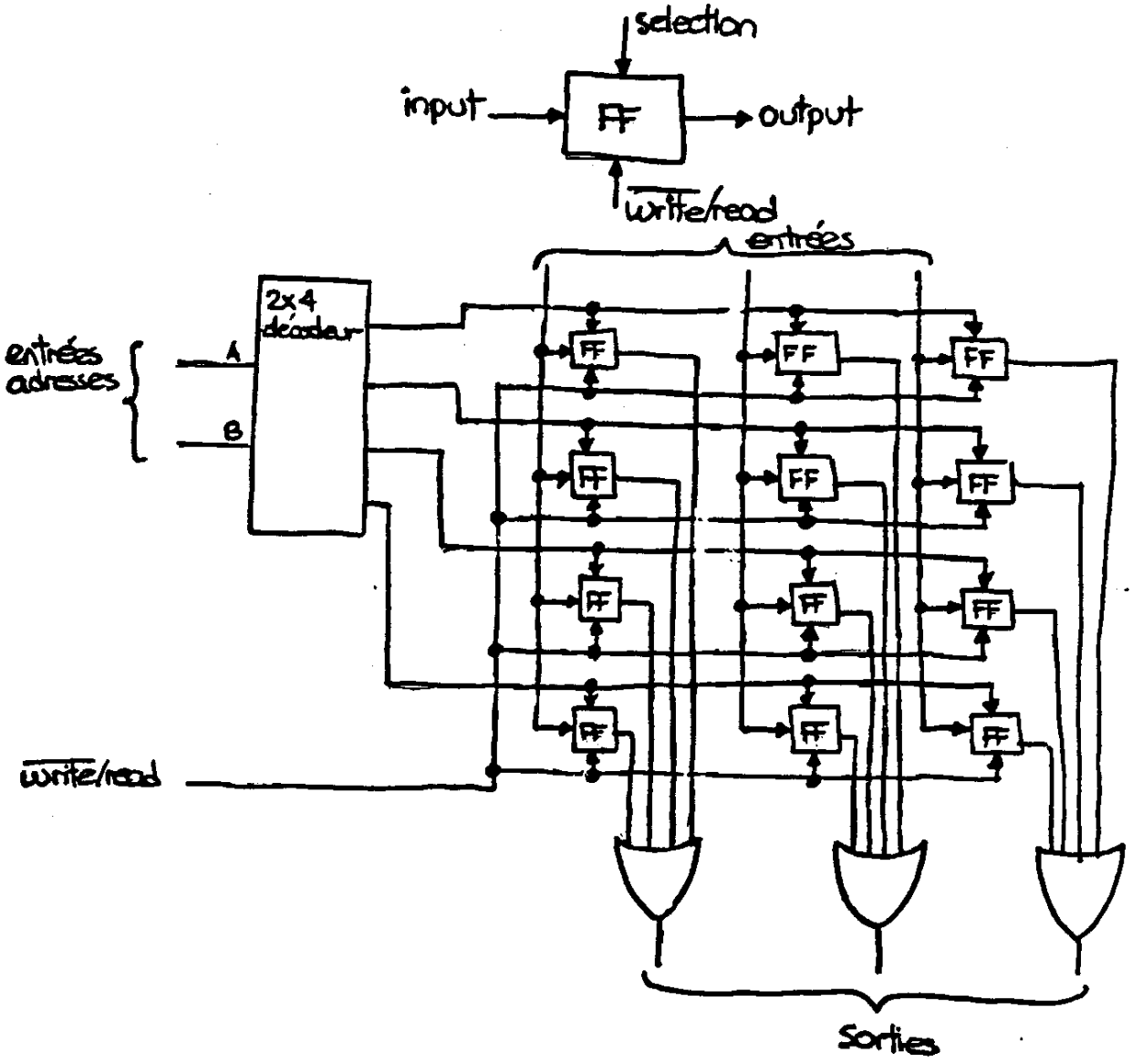


Seules les m bascules sélectionnées par l'adresse ne sont pas dans un état haute impédance.

Principe de base (statique)

la cellule





Capacité : 4 x 3

en dynamique, l'information est retenue par la charge d'un condensateur. Or, le condensateur peut se décharger et ainsi perdre l'information, d'où la nécessité de rafraîchissement.

Les nouvelles mémoires de type VSRAM (Virtually Static RAM) comprennent la circuiterie nécessaire au rafraîchissement et peuvent être considérées comme des SRAM à toutes fins pratiques.

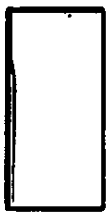
Exemples

dynamique	TMS 4164 (MOS)	64K x 1
	MK 4116	16K x 1
statique	INTEL 2147	4K x 1
	TMS 4016	2K x 8
	7489 (TTL)	64 x 4
	INTEL 8185 (NMOS)	1K x 8

Dominique Grenier

Le 21 octobre 1985

Revision 25 fév. 86 Benoit Nadeau



PAL Introduction

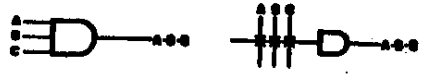


Figure 2

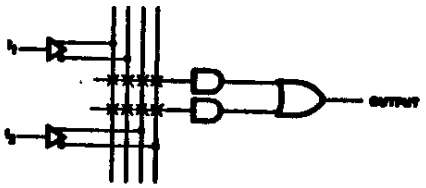


Figure 3

As a simple PAL example, consider the implementation of the transfer function:

$$\text{Output} = I_1\bar{I}_2 + \bar{I}_1 I_2$$

The normal combinatorial logic diagram for this function is shown in figure 4, with the PAL logic equivalent shown in figure 5.

Figure 4

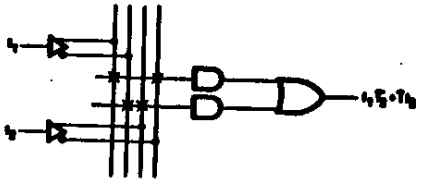


Figure 5

Using this logic convention it is now possible to compare the PAL structure to the structure of the more familiar PROM and PLA. The basic logic structure of a PROM consists of a fixed AND array whose outputs feed a programmable OR array (figure 6). PROMs are low-cost, easy to program, and available in a variety of sizes and organizations. They are most commonly used to store computer programs and data. In these applications the fixed input is a computer memory address; the output is the contents of that memory location.

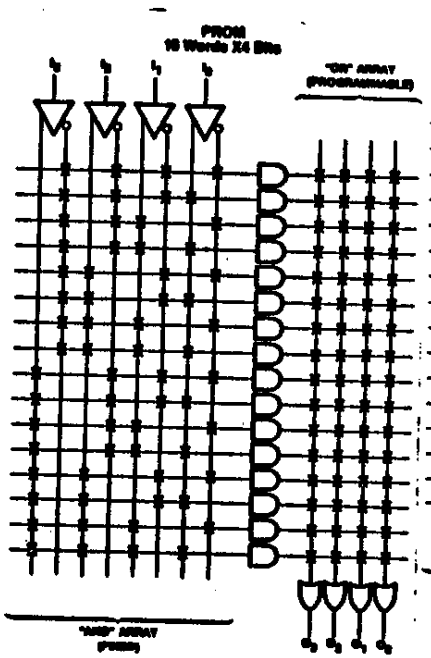


Figure 6

The basic logic structure of the PLA consists of a programmable AND array whose outputs feed a programmable OR array (Figure 7). Since the designer has complete control over all inputs and outputs, the PLA provides the ultimate flexibility for implementing logic functions. They are used in a wide variety of applications. However, this generality makes PLAs expensive, quite formidable to understand, and costly to program (they require special programmers).

The basic logic structure of the PAL, as mentioned earlier, consists of a programmable AND array whose outputs feed a fixed OR array (Figure 8). The PAL combines much of the flexibility of the PLA with the low cost and easy programmability of the PROM. Table 1 summarizes the characteristics of the PROM, PLA, and PAL logic families.

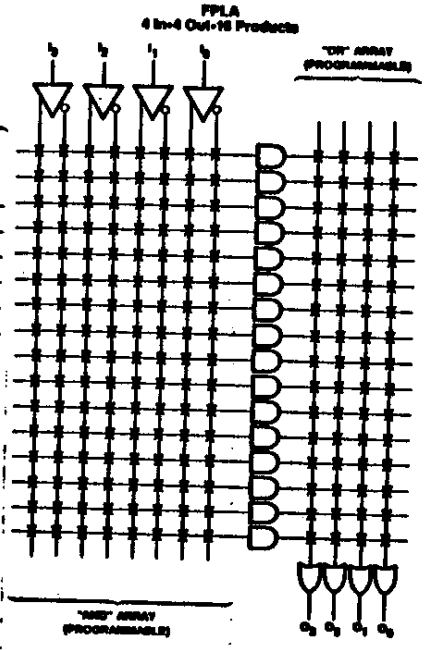


Figure 7

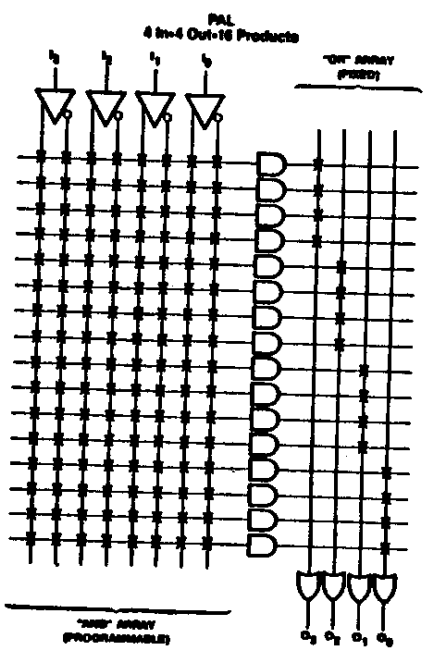


Figure 8

	AND	OR	OUTPUT OPTIONS
PROM	Fixed	Prog	TS, OC
FPLA	Prog	Prog	TS, OC, Fusible Polarity
FPGA	Prog	None	TS, OC, Fusible Polarity
FPLS	Prog	Prog	TS, Registered Feedback, I/O
PAL	Prog	Fixed	TS, Registered Feedback, I/O

Table 1

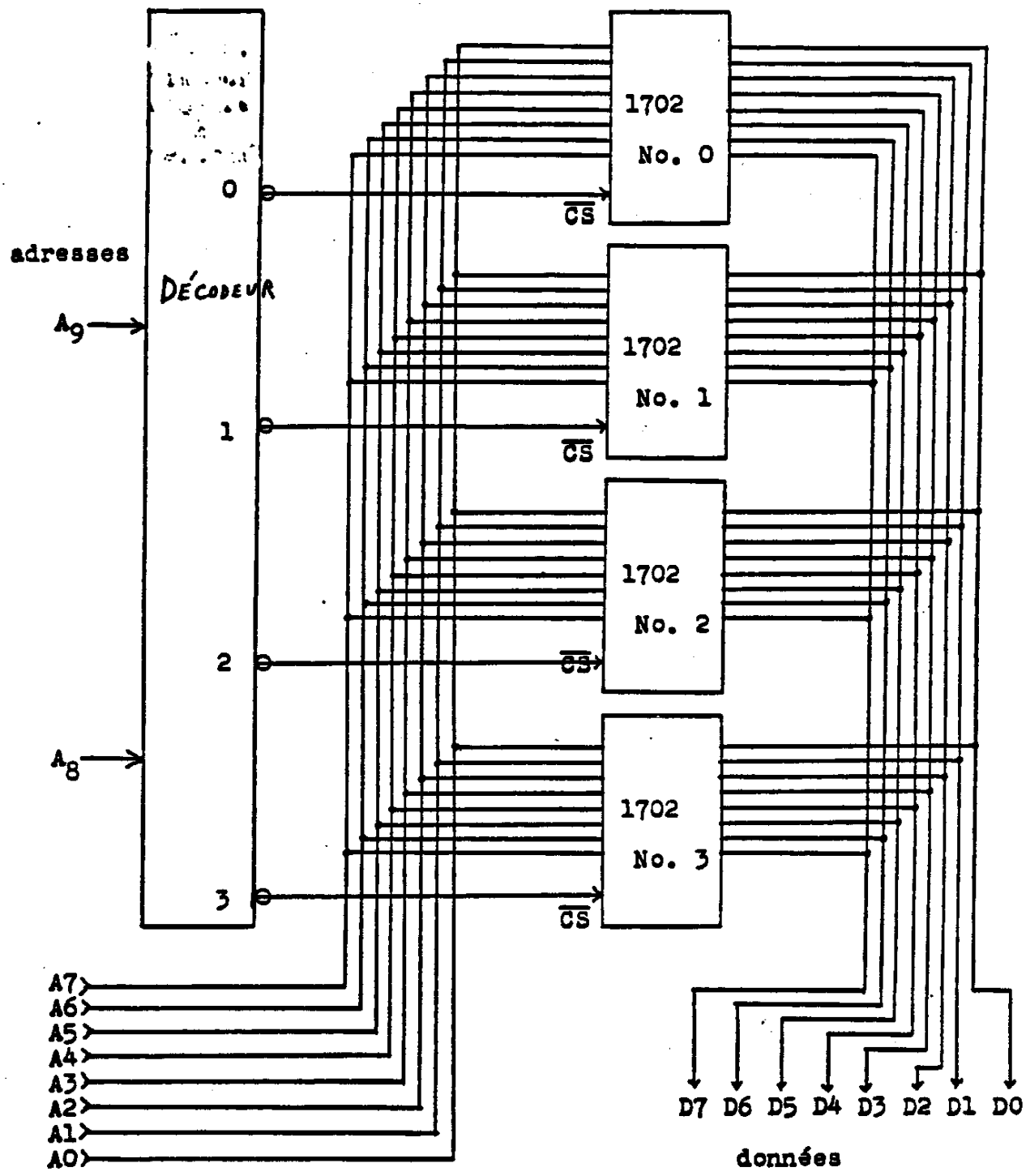


Figure 70b, Assemblage de EPROMs de 256 x 8 bits en une mémoire morte de 1024 x 8 bits.

CHAPITRE 6

LES OPÉRATIONS ARITHMÉTIQUES DE BASE

Le chapitre 1 expliquait comment les nombres pourraient être représentés sous forme binaire. Trois représentations avaient alors été décrites :

- Valeur absolue et bit de signe
- Complément restreint (complément "1")
- Complément vrai (complément "2")

Ce chapitre aborde l'aspect des opérations arithmétiques élémentaires utilisant ces notations. Des circuits logiques pour l'addition, la soustraction, la multiplication sont décrits, et diverses variantes de réalisation sont discutées.

6.1 - Addition binaire avec blocs élémentaires

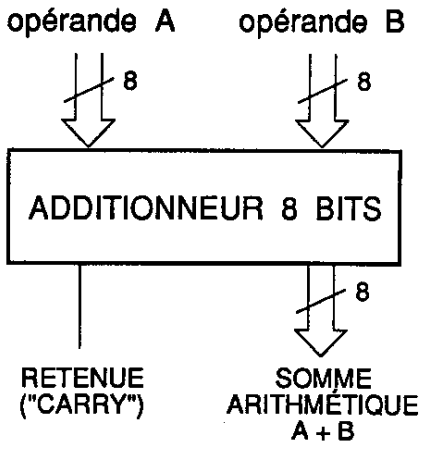
Lorsqu'on aborde le thème des opérations arithmétiques, la signification des niveaux logiques doit d'abord être modifiée. Ainsi, aux chapitres précédents, les niveaux logiques avaient la signification suivante :

(en TTL) + 5 V → "1" logique → "VRAI"
0 V → "0" logique → "FAUX"

Si on veut maintenant réaliser des opérations arithmétiques avec ces niveaux logiques, il faut plutôt leur attribuer la signification suivante :

(en TTL) + 5 V → "1" arithmétique en base 2
0 V → "0" arithmétique en base 2

L'opération arithmétique d'addition de deux nombres de 8 bits en notation binaire est représentée au schéma ci-dessous :



La réalisation matérielle de l'additionneur doit être abordée de façon modulaire. Une approche modulaire implique qu'on se concentre sur la réalisation d'un bloc simple puisque ce bloc est répété de façon plus ou moins régulière pour réaliser complètement la fonction d'addition.

Prenons par exemple le cas de l'additionneur appelé "demi-additionneur" à deux bits (aussi appelé "HALF-ADDER").

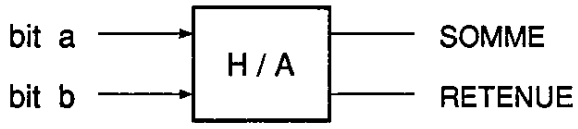
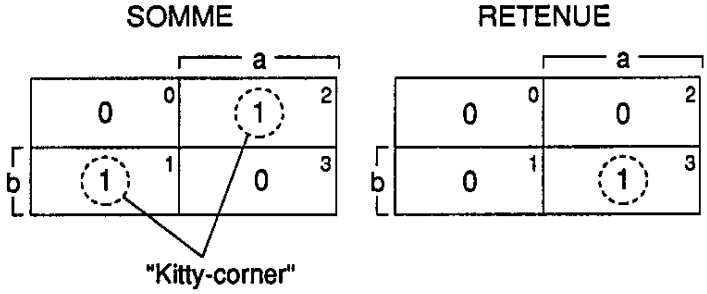


Schéma du "HALF - ADDER" 2 bits.

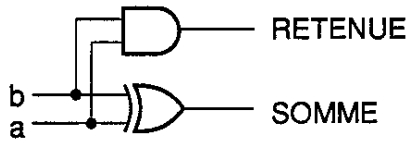
La table de vérité du "HALF-ADDER" équivaut à la somme de 2 nombres d'un seul bit (a et b) avec retenue sans codage donc :

a	b	SOMME	RETENUE
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



SOMME = a ⊕ b RETENUE = a • b

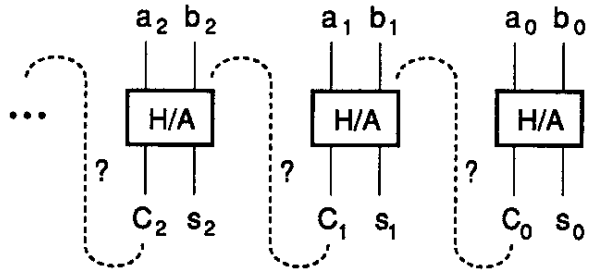
Le schéma logique du "HALF-ADDER" est donc simplement :



H / A 2 bits, nombres sans codage

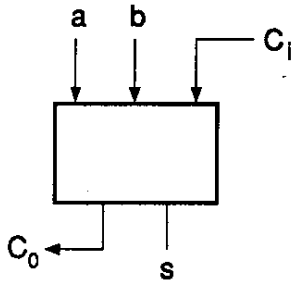
Si on veut utiliser l'additionneur "HALF-ADDER" comme bloc de base pour construire un additionneur à 8 bits par exemple, on constate tout de suite sa faiblesse :

Le "HALF-ADDER" ne peut propager la retenue d'un étage à l'autre d'un circuit l'addition



On constate la nécessité d'un bloc élémentaire plus complet en tenant compte de la retenue provenant des étages d'addition de poids moins significatif. Un bloc élémentaire d'addition satisfaisant ces conditions est l'additionneur complet aussi appelé "FULL-ADDER".

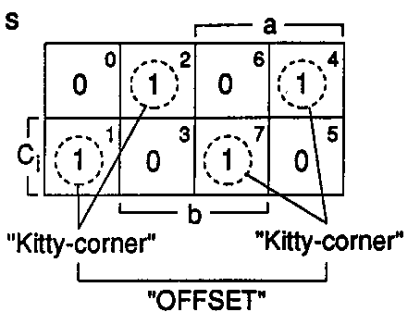
Le schéma de principe du "FULL-ADDER" est :

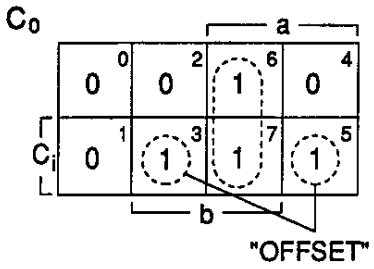


a, b : opérandes
C_i : "carry in"
C_0 : "carry out"

La table de vérité du "FULL-ADDER" est :
(2 bits nombres positifs)

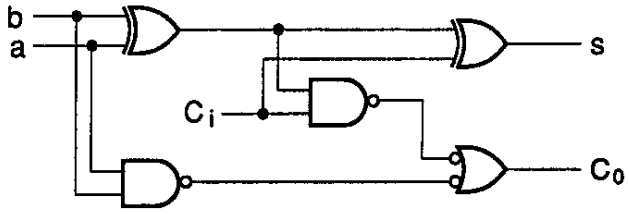
a	b	C_i	s	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





$s = (a \oplus b) \oplus C_i$ $\qquad$ $C_0 = ab + C_i(a \oplus b)$

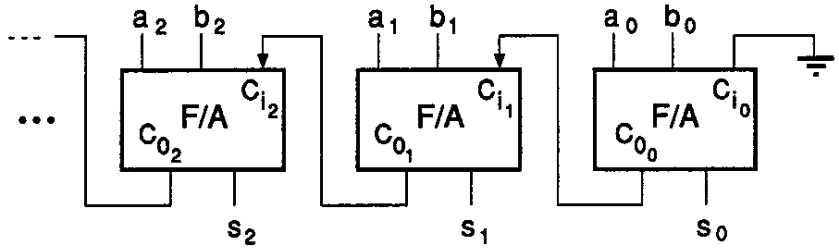
La réalisation du "FULL-ADDER" en ***portes logiques*** *donne donc :*



"FULL-ADDER"

6.1.1 - L'additionneur à retenue décalée ("RIPPLE CARRY")

On peut maintenant utiliser l'additionneur complet de la section précédente pour réaliser un additionneur à plusieurs bits, dit "à retenue décalée".

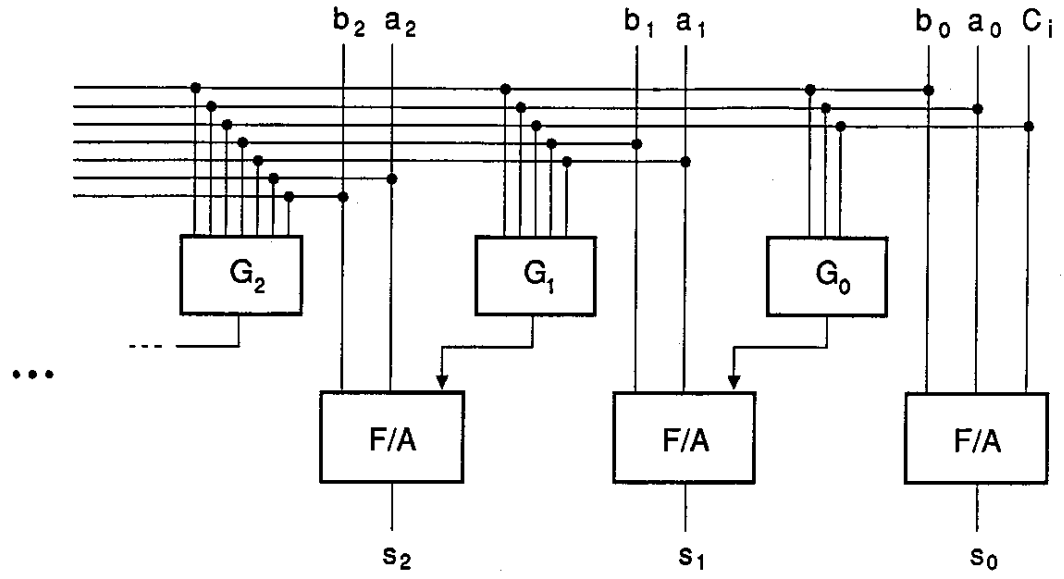


Remarque 1 : L'étage initial peut-être un "HALF-ADDER" puisqu'il n'y a pas de retenue provenant des étages précédents pour le bit de poids faible.

Remarque 2 : On remarque que la somme de l'étage "n" n'est stable que lorsque la retenue provenant de l'étage "(n-1)" est stable. Le fait que la retenue soit décalée d'un étage à l'autre ("RIPPLE") affecte la rapidité globale de l'additionneur. En effet, la retenue finale de la somme ne sera connue qu'après un intervalle de temps proportionnel au nombre d'étages de l'additionneur.

6.1.2 - L'additionneur multiple à retenue anticipée ("LOOK-AHEAD CARRY")

Le schéma ci-dessous représente un additionneur à retenue anticipée :



Ici, la retenue est "anticipée", c'est-à-dire qu'elle est calculée à chaque étage par un bloc générateur de retenue "G_i", tenant compte de toutes les entrées des étages de poids inférieur.

Ainsi, somme et retenue finale apparaissent simultanément à tous les étages, ce qui est un avantage certain sur l'additionneur "RIPPLE CARRY".

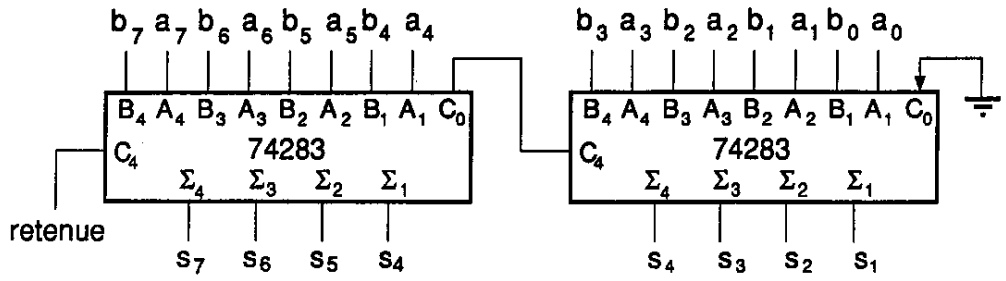
L'inconvénient de l'additionneur "LOOK-AHEAD-CARRY" est qu'il est de conception plus complexe que le "RIPPLE" à cause justement du calcul de la retenue anticipée qui se complique d'ailleurs au fur et à mesure que le poids des bits augmente dans la somme.

6.1.3 - Exemples TTL d'un additionneur 4 bits, le 74283 (7483) et 74181.

Le circuit intégré 74283 est un additionneur à 4 bits de type "FULL-ADDER" à retenue anticipée.

On peut placer plusieurs 74283 en cascades. La retenue est alors anticipée à l'intérieur de chaque 74283 et elle est propagée ("RIPPLE") entre chacun d'eux.

Exemple d'additionneur 8 bits de 2 nombres binaires positifs avec des 74283 :



La retenue d'un 74283 est générée en ≃ 10 µs donc, le résultat est stable sur sur 8 bits après

$$2 \times 10 \simeq 20\ \mu s$$

Le circuit "ALU" ("Arithmetic and Logic Unit) 74181, permet aussi de faire des additions binaires avec anticipation partielle de la retenue. Les additions et autres opérations arithmétiques sont sur 4 bits. Ce circuit est d'une utilisation relativement complexe.

Note : il existe des circuits (ex: 74182) de génération anticipée de la retenue. Ceux-ci peuvent être utilisés avec des 74181 pour obtenir des circuits additionneurs à retenue anticipée complète.

6.2 - Addition et soustraction parallèle.

Lorsqu'on désire concevoir un circuit fonctionnel permettant l'addition et / ou la soustraction, alors il faut ajouter quelques éléments permettant de générer une soustraction à partir d'un circuit additionneur.

Une approche courante est d'utiliser un circuit additionneur avec un circuit générateur du complément vrai. Ainsi la soustraction se résume à l'addition de 2 nombres dont l'un a été complémenté préalablement.

Exemple : sur 3 bits

décimal	complément vrai (binaire)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
-4	1 0 0
-3	1 0 1
-2	1 1 0
-1	1 1 1

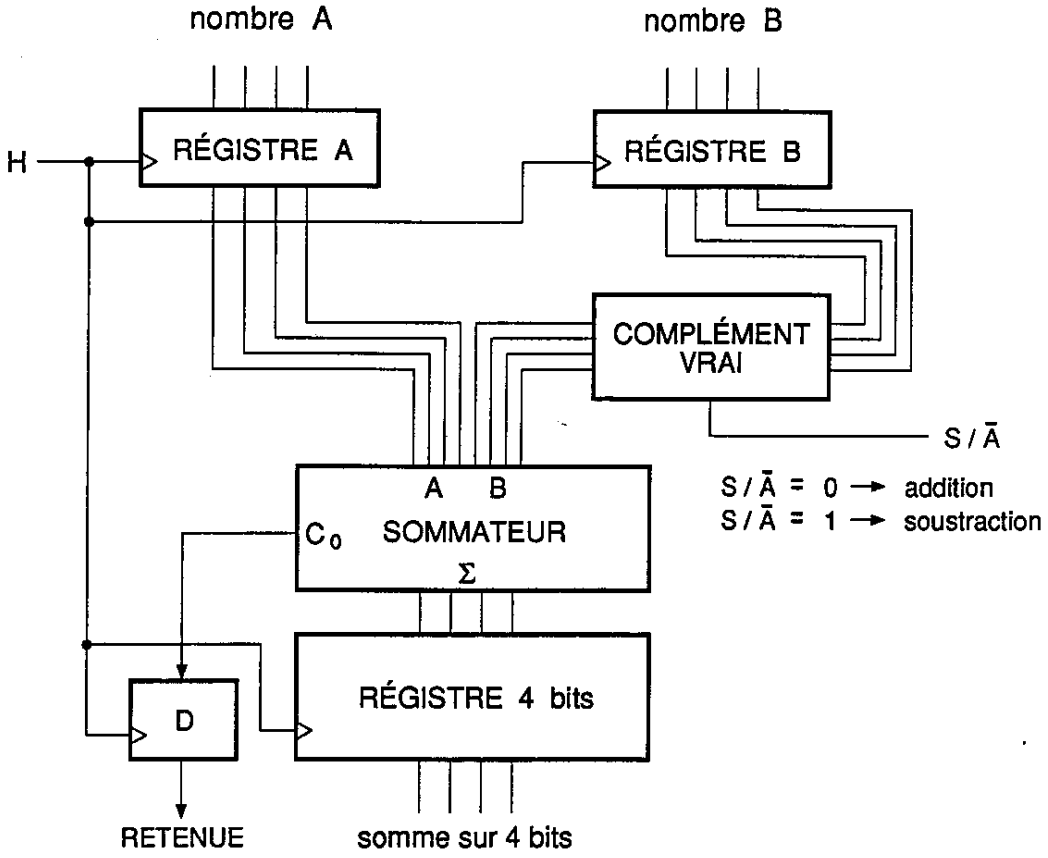
→ $2 - 3 = (2)_{10} + (-3)_{10} : (010)_{CV} + (101)_{CV} = (111)_{CV} = (-1)_{10}$

$3 - 2 = (011)_{CV} + (110)_{CV} = (1\ 001)_{CV} = (+1)_{10}$ si on néglige la retenue.

retenue

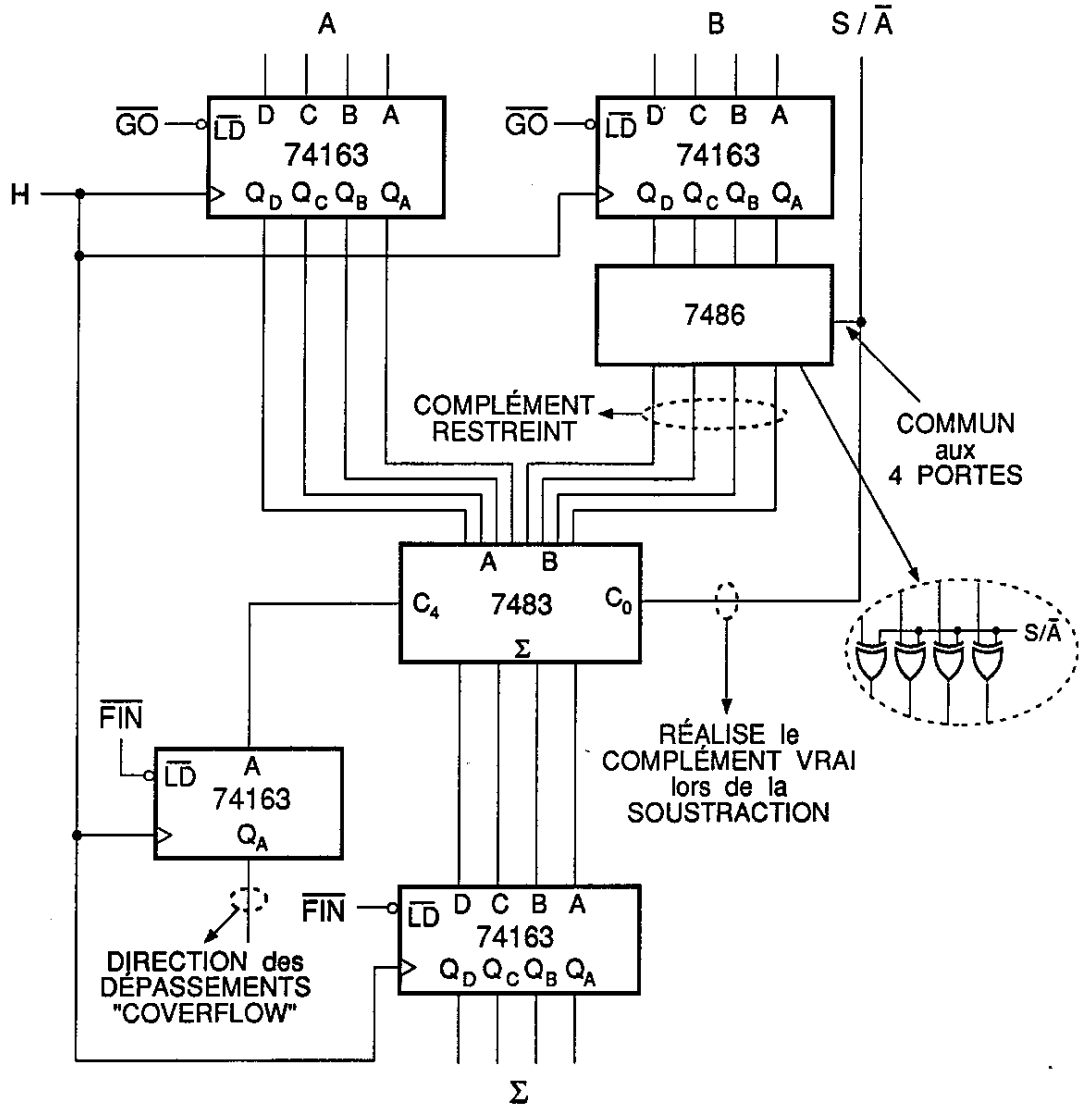
On retrouve en annexe les différents cas possibles pour des opérations arithmétiques selon le type de complément utilisé.

Un schéma additionneur / soustracteur en complément vrai, peut donc prendre l'allure générale suivante :



À remarquer : le sommateur est un circuit purement combinatoire. Il est donc nécessaire de conserver les opérandes et le résultat (somme et retenue) dans les régistres, si on désire les utiliser dans une autre partie d'un circuit.

Une réalisation possible du sommateur parallèle sur 4 bits pourrait être la suivante :



- Le signal "GO" est un blip qui indique que les opérandes sont correctes et le signal "FIN" est un autre blip décalé d'un coup d'horloge.

- Pour que l'addition / soustraction soit correcte, il faut choisir la fréquence de l'horloge de sorte que les circuits aient un délai suffisant pour effectuer l'opération.

- Ce circuit d'addition parallèle est très rapide.

6.3 - Addition et soustraction série.

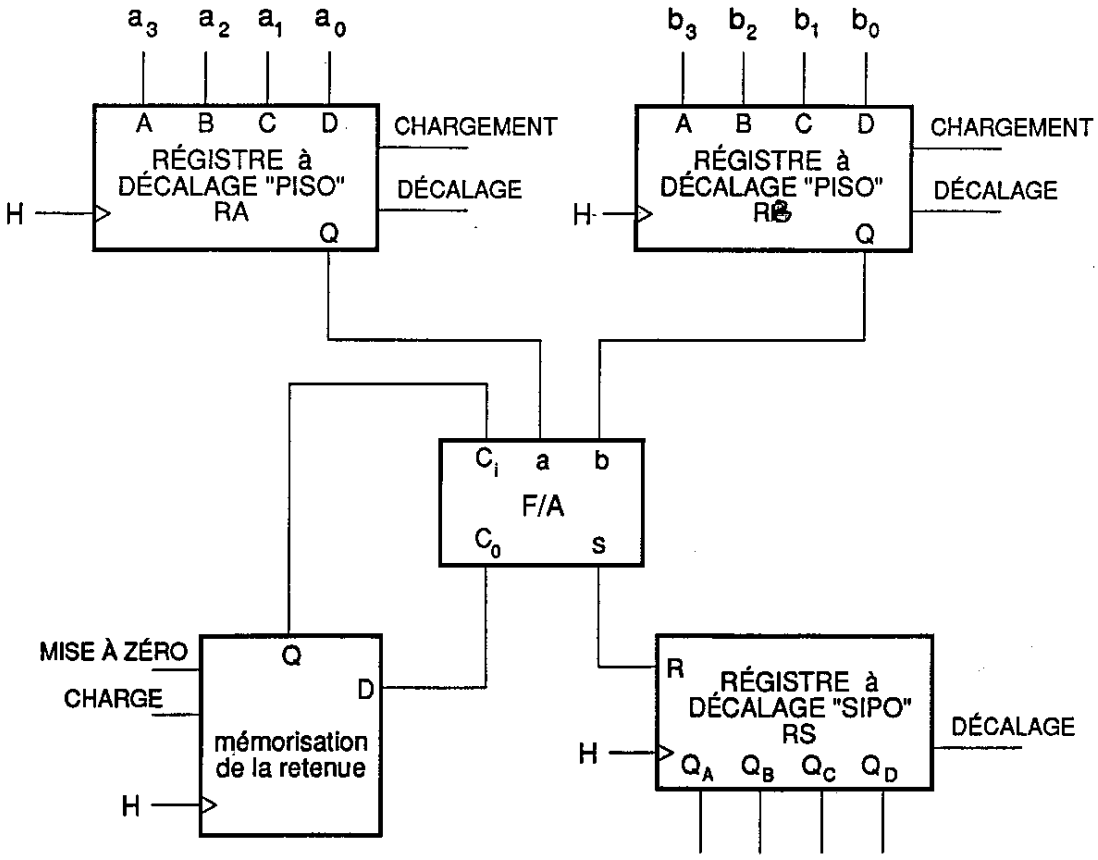
Le circuit d'addition parallèle est très simple de conception mais il est cependant relativement coûteux en nombre de boîtiers.

On remarque cependant que l'itération dans l'espace de l'addition parallèle, peut être remplacée par une itération dans le temps : c'est l'additionneur série.

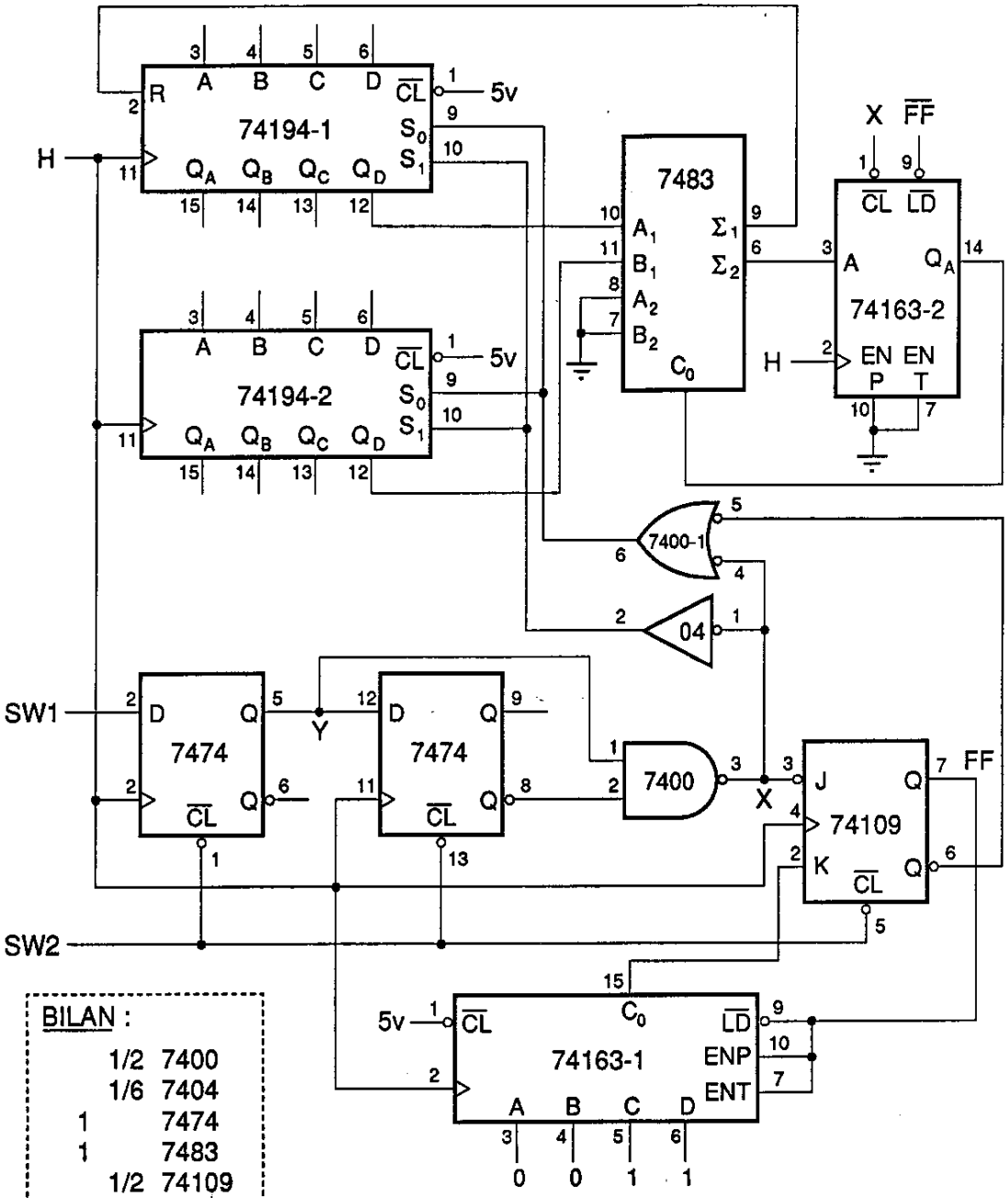
L'additionneur série est plus compliqué dans son fonctionnement. De nos jours l'additionneur serie devient de moins en moins intéressant.

Le schéma de principe de cette page donne une idée des blocs de base qui sont nécessaires à sa réalisation.

Schéma de principe : Additionneur, mots de 4 bits.
3 registres à décalage RA, RB, RS.



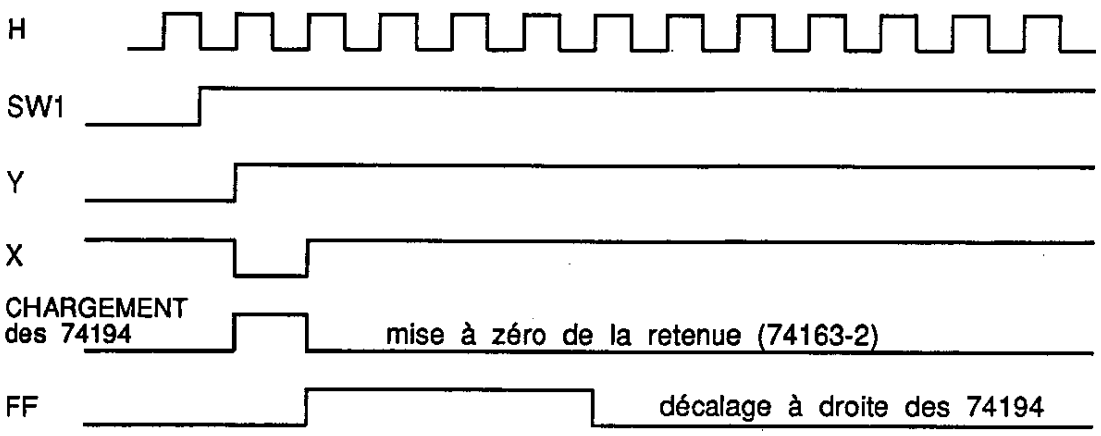
(L'addition série...)



BILAN :		
	1/2	7400
	1/6	7404
1		7474
1		7483
	1/2	74109
2		74163
2		74194
7	1/6	boitiers

S_1	S_0	Q_A^+	Q_B^+	Q_C^+	Q_D^+
0	0	Q_A^-	Q_B^-	Q_C^-	Q_D
0	1	R	Q_A^-	Q_B^-	Q_C^-
1	1	A	B	C	D

Pour mieux comprendre le principe de fonctionnement de l'additionneur série présenté, voici le diagramme temporel des signaux :



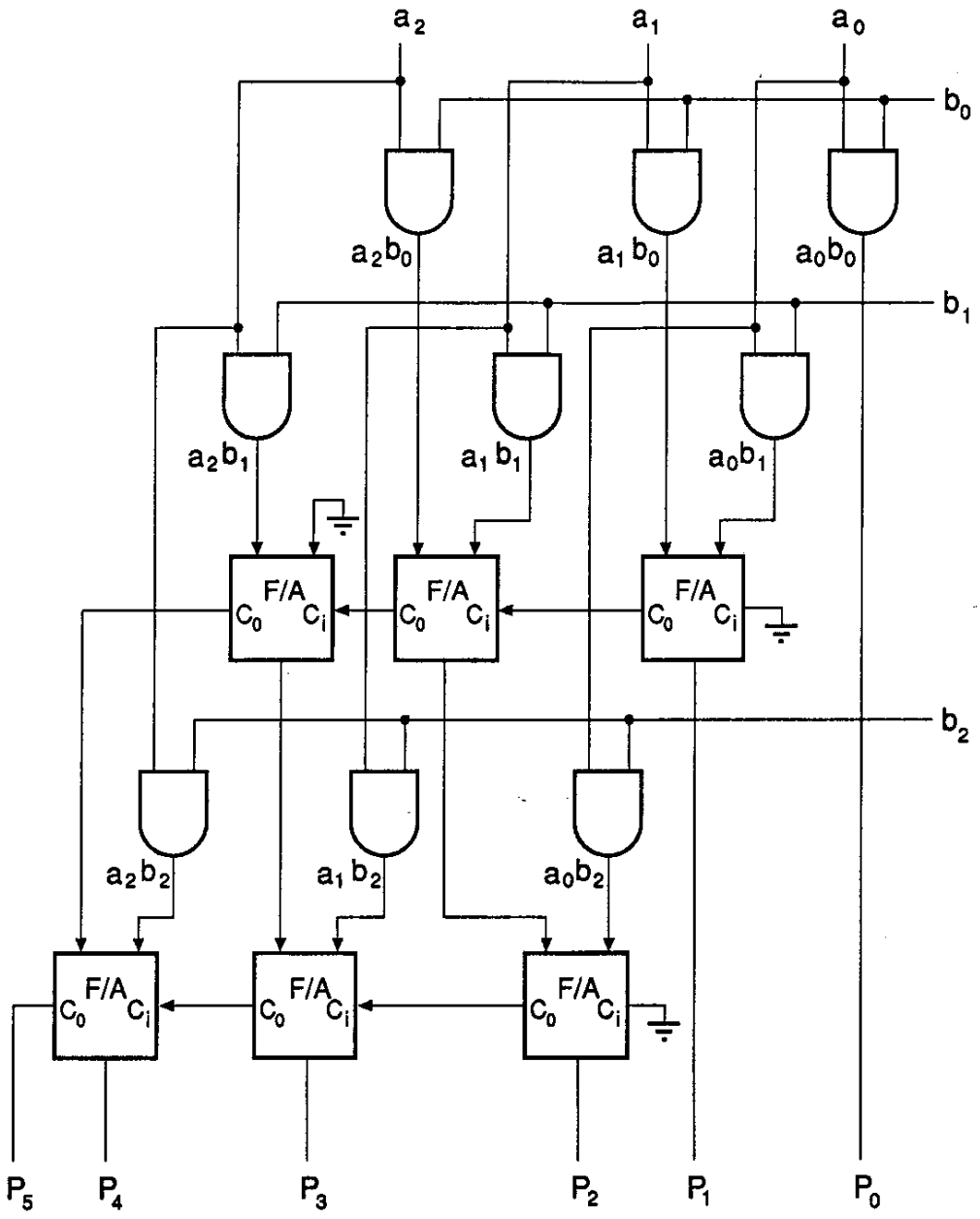
6.4 - La multiplication parallèle.

La multiplication peut-être une opération complètement combinatoire. On peut la réaliser en parallèle avec des portes "ET" et des "FULL ADDERS"

Soit la multiplication de 2 nombres positifs codées sur 3 bits. On a :

			a_2	a_1	a_0
			b_2	b_1	b_0
			$a_2 b_0$	$a_1 b_0$	$a_0 b_0$
		$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	
retenue	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$		
P_5	P_4	P_3	P_2	P_1	P_0

Cette structure suggère immédiatement la réalisation parallèle suivante utilisant des "FULL-ADDERS" et des portes "ET".



3 bits x 3 bits ⟶ 2 étages de sommations et
3 étages de produits

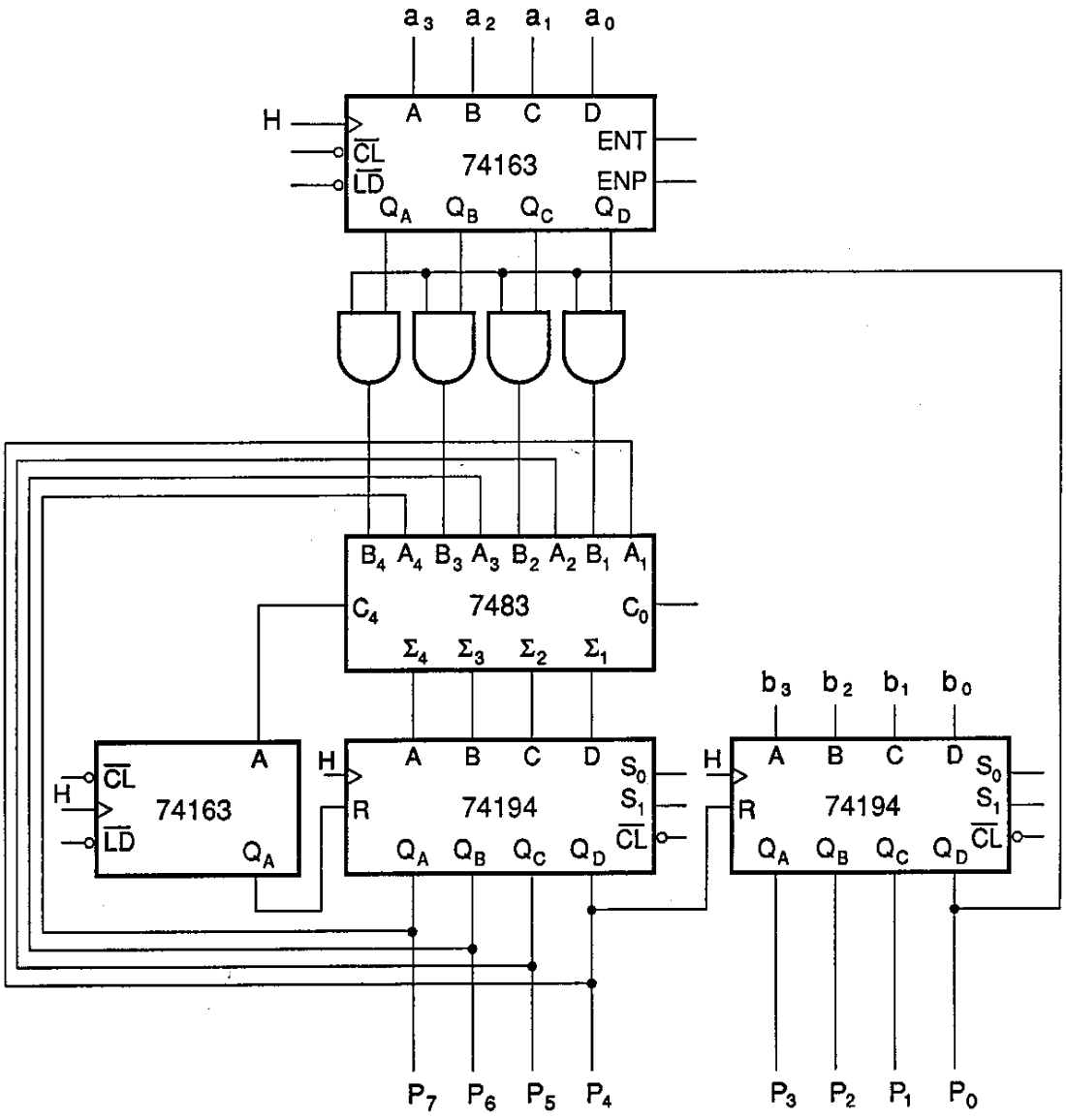
Schéma de principe de multiplicateur parallèle.
(circuit entièrement combinatoire)

Un exemple TTL d'un multiplicateur 4 bits (binaire) est le 74284.

6.5 - Multiplicateur parallèle-série.

On peut remplacer une partie de parallélisme spatial du circuit de multiplication parallèle par une séquence d'opérations répétées dans le temps, sous la commande d'une horloge. Le circuit de multiplication ainsi obtenu n'est plus entièrement combinatoire, mais requiert plutôt un ensemble de régistres (à décalage et parallèle).

Un circuit partiel (sans commande) de multiplication pourrait prendre la forme suivante :



6.6 - "LOOK-UP TABLES" de multiplication

On peut programmer un "ROM" pour que, de deux opérandes de 4 bits, la sortie sur 8 bits corresponde à la multiplication de ces deux nombres.

Ainsi une multiplication 4 bits par 4 bits, exigerait un "ROM" de :

$2^4 \times 2^4 = 2^8 =$ 256 octets (1 octet = 8 bits)

Tandis qu'une multiplication 8 bits par 8 bits demande :

$2^8 \times 2^8 =$ 65536 octets

On dépasse donc rapidement la capacité d'un "ROM" standard. On peut cependant combiner des "ROMs" et des "FULL-ADDERS" et réussir une multiplication sur 8 bits :

Soit X8 et Y8 les deux nombres à multiplier (8 bits chacun)

on peut écrire que :

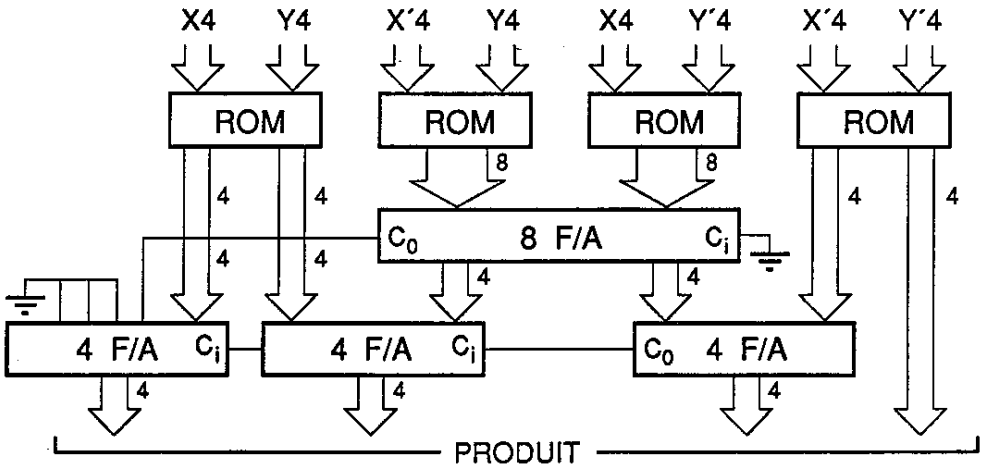
$X8 = X4 + X'4$
$Y8 = Y4 + Y'4$

où { X4, Y4 sont les 4 MSB de X8 et Y8 respectivement
{ X'4, Y'4 sont les 4 LSB de X8 et Y8 respectivement

On démontre alors que :

$$X8 \cdot Y8 = X4\,Y4 + X'4\,Y4 + X4\,Y'4 + X'4\,Y'4$$

Le circuit de principe réalisant la multiplication devient :



EXEMPLE : X8 = 1001 1101 → $(157)_{10}$
Y8 = 0111 0111 → $(119)_{10}$

X4 : 1001 X'4 : 1101
Y4 : 0111 Y'4 : 0111

```
X'4 Y'4:                  0 1 0 1 1 0 1 1
X'4 Y4 :          0 1 0 1 1 0 1 1
X4  Y'4:          0 0 1 1 1 1 1 1
X4  Y4 :  0 0 1 1 1 1 1 1
          ---------------------------------
          0 1 0 0 1 0 0 0 1 1 1 1 1 0 1 1    (18683)10
```

CHAPITRE 7

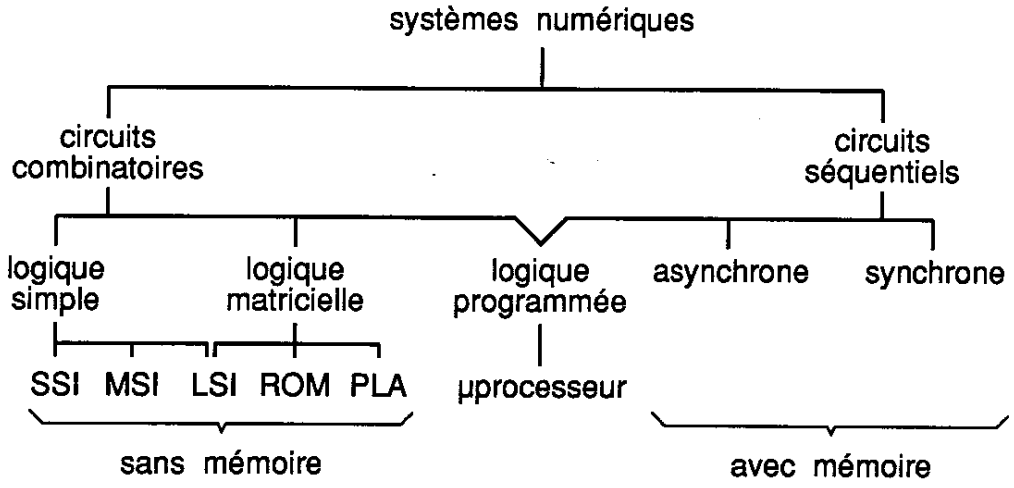
CIRCUITS SÉQUENTIELS SYNCHRONES

7.1 - Introduction

7.1.1 - Définition

Il existe plusieurs sortes de problèmes auxquelles s'appliquent les techniques numériques. Or l'une des plus importantes applications réside dans la commande, où les signaux reçus et générés par le système numérique commandent une machine quelconque (ou un procédé) (des éléments fonctionnels ou des systèmes analogiques électriques ou mécaniques...) selon une séquence bien déterminée. Ce type d'application ne peut être réalisé à partir de circuits combinatoires seulement car les signaux générés doivent dépendre des conditions d'entrée présentes et de l'histoire passée du système numérique.

En résumé, un circuit séquentiel est un circuit dont les états de la sortie se suivent selon une séquence prédéterminée par l'état présent et les commandes d'entrée.



La différence fondamentale entre le synchrone et l'asynchrone se situe au niveau du synchronisme de l'action: celle-ci étant enclenchée seulement sur l'ordre de l'horloge dans un circuit synchrone. L'horloge étant absente dans les circuits asynchrones.

7.1.2 - Classes de machine synchrone

L'ensemble des machines séquentielles synchrones se divise principalement en 3 classes.

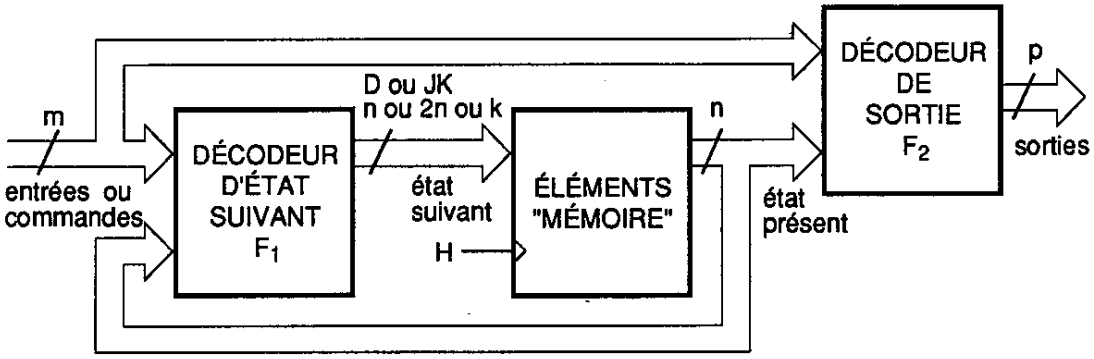
classes → A : machine de Mealy
B : machine de Moore
C : machine de Moore sans décodeur de sortie

Ces classes, en fait, se subdivisent en plusieurs niveaux de hiérarchie selon l'utilisation de plus en plus fréquente des circuits à haut niveau d'intégration. Autrement dit, la structure classique de départ se transforme peu à peu avec l'emploi de multiplexeurs, ROM ou PLA.

machine de Mealy : les sorties sont fonction de 2 ensembles de variables ⟶ les conditions d'entrée présentes ⟶ l'état présent

machine de Moore : les sorties sont fonction strictement de l'état présent de la machine.

La machine de Mealy est, bien sûr, plus générale et c'est pour cette raison que la synthèse présentée dans ce chapître, est celle aboutissant à une structure de Mealy. Il est aussi bon de noter qu'une machine de Moore sans décodeur de sortie suffit à la synthèse d'un compteur.



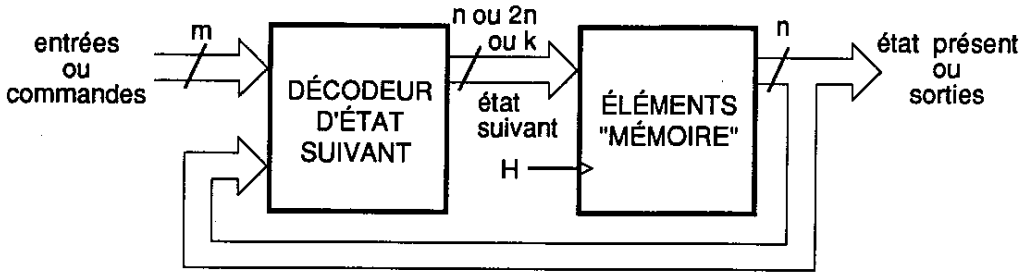
STRUCTURE DE MEALY :

* *F_1 et F_2 sont des circuits combinatoires classiques (SSI ou MSI) ou avec ROM ou PLA dans les circuits de grande complexité.*
* *Les éléments "mémoire" sont constitués des bascules D ou J-K "edge-triggered" pour un meilleur fonctionnement ou de compteurs ou de registres. Tous les éléments "mémoire" sont actionnés d'une façon synchrone i.e. horloge commune.*
* *Le nombre maximal d'états possibles correspond à 2^n ou n représente le nombre de sorties des éléments de "mémoire".*
* *Les n sorties des éléments "mémoire" sont les bits de codage de l'état présent.*
* *Le décodeur d'état suivant prépare les commandes à effectuer pour établir la transition vers l'état suivant au prochain coup d'horloge. Le décodeur possède n sorties si les éléments "mémoire" sont des bascules D ou 2n sorties pour des bascules J-K ou k sorties dépendamment du nombre de commandes externes synchrones des registres / compteurs.*

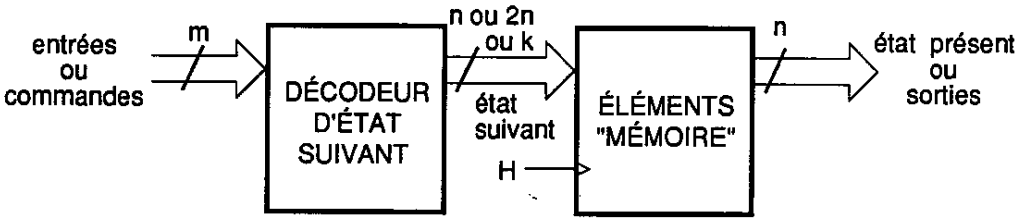
décodeur d'état suivant :

n sorties ⟶ bascules D (entrées D)
2n sorties ⟶ bascules J-K (entrées J, entrées K)
k sorties ⟶ régistres ou compteurs

* *Les m entrées ou commandes gèrent la marche à suivre lorsque plusieurs transitions sont possibles.*



structure de Moore sans décodeur de sortie



structure "look up memory"

7.2 - Règles de synchronisme

La fiabilité d'un circuit séquentiel synchrone est quelque chose de bien connue dans le monde du numérique. Cette fiabilité, cependant, dépend de l'application méthodique d'un certain nombre de règles ayant trait au synchronisme qui est à la base du mode synchrone.

Contrairement aux désigns asynchrones (tels que vus dans un chapître subséquent), les circuits séquentiels synchrones éliminent les risques de course critique d'oscillation et, par dessus tout, l'importance des aléas notamment les aléas statiques. Mais, ce dernier point requiert, de la part de l'ingénieur, un suivi respectueux des règles de synchronisme.

1. Les seuls éléments "mémoire" sont les bascules D et/ou J-K et/ou régistres/compteurs activées par la même horloge.

2. Toutes les entrées externes ou commandes externes doivent être amorties puis synchronisées sur l'horloge commandant les éléments "mémoire". L'amortissement retire les aléas dynamiques qui pourraient avoir un effet provoquant des transitions non-valides.

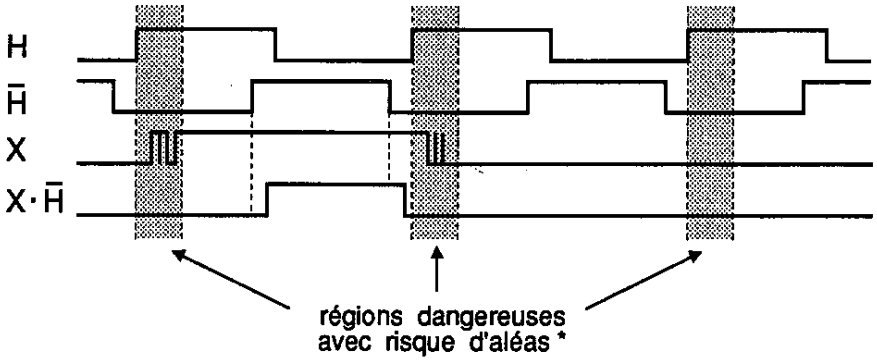
3. Ne jamais utiliser un signal synchrone ou une combinaison issue de signaux synchrones avec l'horloge elle-même dans le but de produire un signal de synchronisme dans une partie du circuit.

   Les problèmes naissent du fait que

   - les signaux synchrones risquent d'être porteurs d'aléas statiques immédiatement après la montée de l'horloge.
   - les signaux synchrones sont légèrement retardés par rapport à l'horloge.

   Si un autre signal de synchronisme est absolument nécessaire, il faut jouer prudemment: multiplier le signal synchrone par $\overline{H}$ en avance sur H et se servir du résultat avec réserve. En effet, bien que le signal résultant soit exempt d'aléa, il n'est cependant plus sunchronisé sur l'horloge mère du système, H.

   Exemple : X est un signal synchrone avec aléas statiques

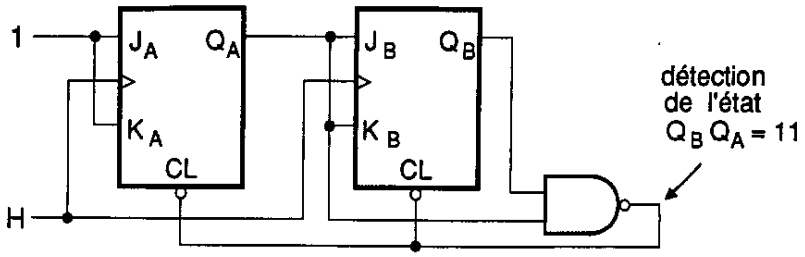


   On remarque que le produit $X \cdot \overline{H}$ ne possède pas d'aléa. Le principe consiste à garder $\overline{H}$ au niveau bas pendant les temps ou il y a risque d'aléas. De plus, même en prenant un signal synchrone non souillé, il n'en demeure pas moins en retard sur la montée de H. Prendre X comme commande de synchronisme entrainerait bien des soucis du genre : une commande synchrone est-elle arrivée avant ou après la montée de X étant donné qu'elle est synchronisée elle aussi par H (donc légèrement en retard sur H mais où se situe-t'elle par rapport à X)?

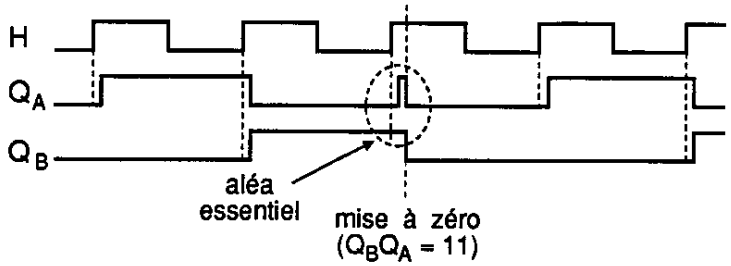
4. Ne jamais utiliser les entrées asynchrones des éléments synchrones sauf pour l'initialisation. L'entrée asynchrone agissant instantannément, le décalage entre l'horloge et le signal synchrone branché sur l'entrée asynchrone suffit pour produire un aléa essentiel.

   Dans un cas extrême, on peut toujours solutionner le problème en se rappelant que l'utilisation de $\overline{H}$ en avance sur H peut éliminer les aléas.

Exemple :

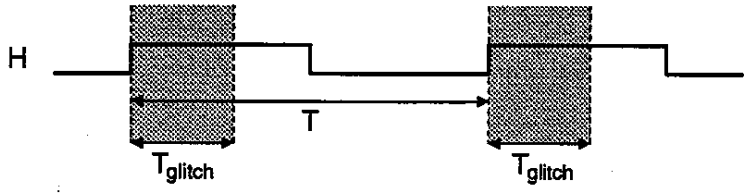


il s'agit d'un compteur binaire 2 bits à 3 états. Lorsqu'on détecte l'état $Q_B Q_A = 11$, on remet immédiatement le compteur à $Q_B Q_A = 00$.



7.2.1 - Régions à risque d'aléa

Précédemment, on a indiqué que les systèmes synchrones sont immunisés contre les aléas statiques. La raison est que les aléas apparaissent lorsque les variables changent. Or, dans un système synchrone, le changement des variables ne s'effectue que sur l'ordre de l'horloge. Il s'ensuit que les aléas sont donc présents immédiatement après la montée de l'horloge mais ils ont disparu au coup d'horloge suivant. La période critique est, pour cela, celle suivant la montée d'horloge et la longueur de cette période dépend du délai de propagation à travers les éléments "mémoire" et le décodeur d'état suivant. Comme la fréquence maximale d'opération est choisie en tenant compte de ce délai de propagation (et donc de la période critique), la longueur de la région à risque d'aléa est inférieure à une pépériode d'horloge, d'où la disparition des aléas à la montée de l'horloge suivante.



$T_{glitch} = \Delta T_{max} = \sum_i \Delta T_i$ (somme des délais de propagation à travers les éléments "mémoire" et le décodeur d'état suivant).

$T > \Delta T_{max}$ (délai de propagation maximal à travers le décodeur d'état suivant et les éléments "mémoire").

7.3 - L'analyse formelle

L'analyse d'un circuit séquentiel synchrone se réalise de façons diverses : chaque individu peut développer sa méthode bien personnelle, sauter des étapes ou reformer les tables. Dans le chapître IV, nous avions démontré que, quelle que soit la méthode, celle-ci se devait d'être méthodique.

On montre ici, les étapes de l'analyse formelle :

a) *Identifier les éléments fonctionnele i.e. les modules de la machine de Mealy :*
   - *décodeur d'état suivant*
   - *éléments "mémoire"*
   - *décodeur de sortie*

b) *Écrire les expressions booléennes pour chacune des commandes des éléments "mémoire" (D ou J et K ou autres).*

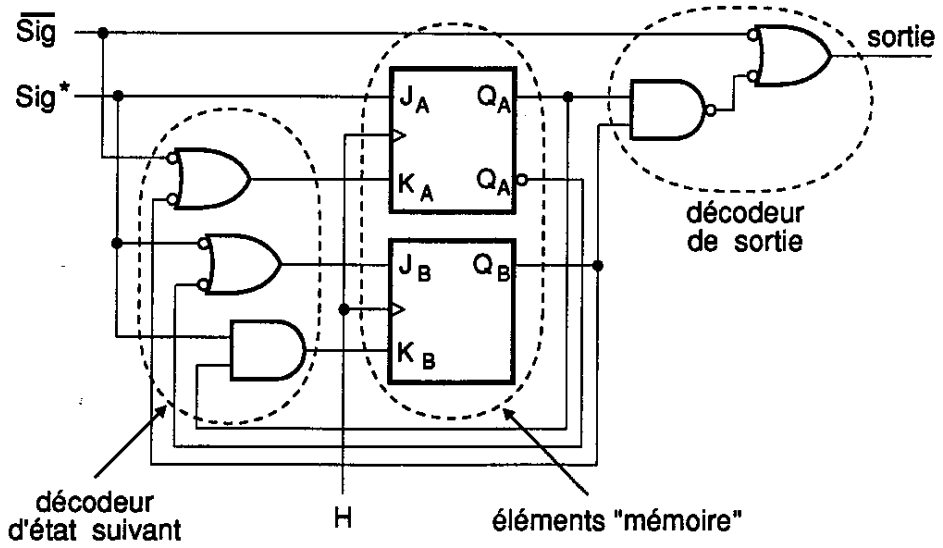
c) *Remplir les tables de Karnaugh qui produisent les mêmes expressions simplifiées qu'en (b).*

d) *À partir de ces tables, déterminer l'action entreprise par les éléments "mémoire" dans chacun des états. Remplir une "table de Karnaugh" pour chacun des éléments "mémoire" dans laquelle on inscrit l'action dépendamment de l'état.*

e) *Dresser une table d'état présent/suivant.*

f) *Dessiner le diagramme d'état (ou graphe) connaissant tous les renseignements utiles :*
   - *transitions (entre les états) conditionnelles aux commandes externes.*
   - *définition de la valeur de la sortie possédant l'équation du décodeur de sortie.*

Exemple :



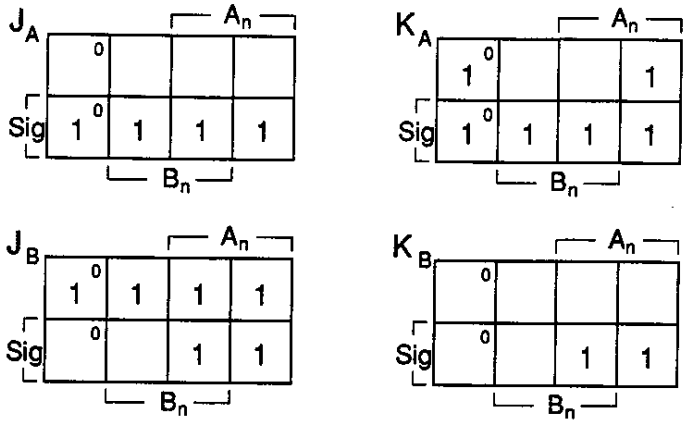
* Sig est un signal synchrone

$J_A = Sig$

$K_A = Sig + \overline{Q}_B \quad (Sig + \overline{B_n})$

$J_B = \overline{Sig} + Q_A \quad (\overline{Sig} + A_n)$

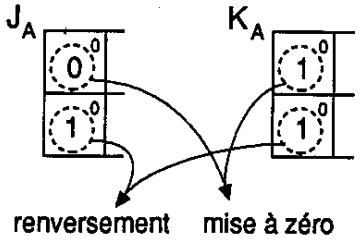
$K_B = Sig \cdot Q_A \quad (Sig \cdot A_n)$

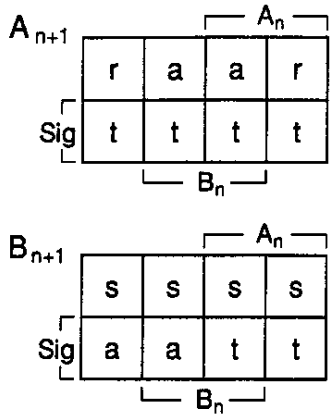


Pour rechercher l'action entreprise, il est nécessaire de connaître la table caractéristique de l'élément "mémoire" employé.

J	K	action	notation
0	0	aucune	a
0	1	mise à zéro	r ("reset")
1	0	mise à un	s ("set")
1	1	renversement	t ("toggle")

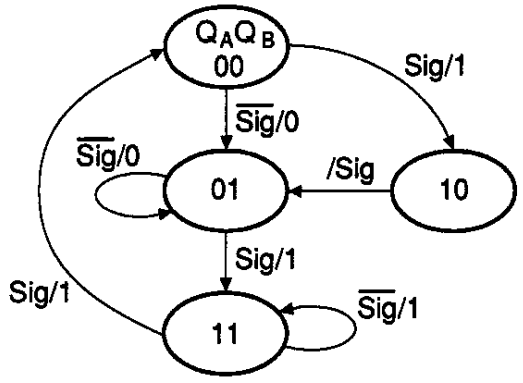
Ainsi, lorsque le système se trouve dans l'état "0" ($Q_A Q_B = 00$), 2 conditions sont possibles : Sig = 0 ou Sig = 1. Dans le premier cas, la bascule A effectuera une mise à zéro à la prochaine montée de H tandis que la bascule B exécutera une mise à un. Dans le second cas, la bascule A renversera pendant que la bascule B demeurera dans son état à la prochaine montée de H.





état présent Q_A Q_B	Sig	état suivant Q_A Q_B	out
0 0	0	0 1	0
	1	1 0	1
0 1	0	0 1	0
	1	1 1	1
1 0	0	0 1	0
	1	0 1	1
1 1	0	1 1	1
	1	0 0	1

$out = Q_A \cdot Q_B + Sig \quad (A_n \cdot B_n + Sig)$



7.4 - Synthèse classique

7.4.1 - Étapes

Tout comme l'analyse, la synthèse d'un circuit séquentiel synchrone exige une méthodologie rigoureuse bien que, selon l'expérience, certaines étapes peuvent être retirées, simplifiées ou modifiées.

Il est important de noter que le décodeur d'état suivant prépare pendant l'état présent de la machine l'action à entreprendre pour changer les sorties des éléments "mémoire" conformément au diagramme d'état. Ce nouvel état doit être prêt lorsque survient la prochaine montée de l'horloge. Il faut donc que les aléas soient terminés → la fréquence de l'horloge doit être choisie en fonction des délais de propagation.

Les étapes formelles de la synthèse d'un circuit séquentiel synchrone sont les suivantes :

a) *Recevoir les spécifications. On obtient ainsi la définition du problème sous la forme d'un diagramme d'état ou d'un diagramme de fonctionnement, ou simplement sous forme qualitative.*

b) *Définir, dans le cas où la définition du problème n'apporte pas un diagramme d'état, un graphe primitif basé sur les informations obtenues. Le graphe primitif se construit en créant un état pour chaque condition de branchement. Il contient donc, possiblement, des états redondants.*

c) *Dresser une table d'état présent/suivant primitive ou une table de transition primitive.*

d) *Repérer systématiquement les états redondants puis réduire les états selon les théorèmes d'états équivalents.*

e) *Faire le codage des états si celui-ci n'est pas spécifié. Le choix du codage influence le coût et la quantité de câblage dans l'implantation du circuit du décodeur d'état suivant et du décodeur de sortie. On peut par exemple choisir entre certains codes déjà vus au chapitre 4 : code binaire, en anneau ou de Johnson.*

f) *Dresser la table d'état/présent ou de transition à partir du graphe simplifié obtenu à la fin de l'étape (d).*

g) *Développer les tables de Karnaugh pour la préparation de l'état suivant et ce à chacune des commandes des éléments "mémoire" choisis. Il est donc nécessaire de se rappeler la table d'excitation de la bascule. Dériver pour des bascules D et J-K en vue de faire la sélection optimale (coût minimum) de la combinaison d'éléments "mémoire" favorisant l'implantation. À cette étape, on obtient la synthèse du décodeur d'état suivant.*

h) *Développer la table de Karnaugh pour définir le décodeur de sortie.*

7.4.2 - Réduction des états

Tout procédé de synthèse doit considérer le problème de minimisation du coût du circuit final. Or, les étapes de départ de la synthèse classique laissent place à une réduction des états. Cette réduction est nécessaire lorsque l'ingénieur reçoit les spécifications du circuit non développé. Il dresse alors un graphe en créant un état pour chaque condition de branchement. Il y a donc risque d'états redondants qu'il faut éliminer par une analyse plus approfondie du problème et par l'usage d'une règle de réduction.

Définition : *2 états, p et q, sont équivalents lorsqu'ils exécutent les mêmes transitions pour toutes les combinaisons possibles des commandes externes.*

(p EQ q) $\Rightarrow$ pour une même combinaison des commandes externes, le circuit séquentiel suit une transition partant de "p" l'amenant vers un même état que la transition suivie en partant de "q"; ce pour l'ensemble des combinaisons.

$\Rightarrow$ pour une même combinaison des commandes externes, les spécifications de sortie sont identiques lorsque le circuit séquentiel est dans l'état "p" ou "q"; ce pour l'ensemble des combinaisons.

Règle: *Si 2 états sont équivalents alors l'un des deux états (n'importe lequel) est redondant.*

état présent	commande C	état suivant	sortie Y
⋮	⋮	⋮	⋮
p	0	r	y_0
	1	s	y_1
⋮	⋮	⋮	⋮
q	0	r	y_0
	1	s	y_1

Exemple :

état présent	C	état suivant	sortie Y
a	0	b	0
	1	c	1
b	0	c	0
	1	a	1
c	0	d	1
	1	e	0
d	0	c	0
	1	a	1
e	0	d	0
	1	c	1

b EQ d

état présent	C	état suivant	sortie Y
a	0	b	0
	1	c	1
b	0	c	0
	1	a	1
c	0	b	1
	1	e	0
e	0	b	0
	1	c	1

a EQ e

remarquez que l'état "d" est devenu l'état "b"

état présent	C	état suivant	sortie Y
a	0	b	0
	1	c	1
b	0	c	0
	1	a	1
c	0	b	1
	1	a	0

remarquez que l'état "e" est devenu l'état "a"

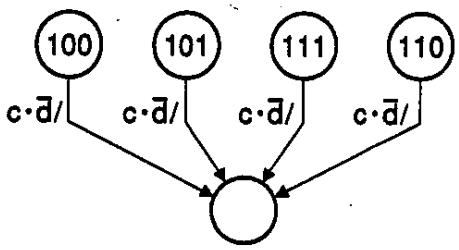
Dans la première table état présent/suivant, la réduction de l'état "a" et "e" n'est pas mise en évidence. Il faut donc s'efforcer d'extraire toutes les redondances d'états possibles en dressant de nouvelles tables.

7.4.3 - Codage

Contrairement aux circuits séquentiels asynchrones, aucune loi ne régit le codage des circuits séquentiels synchrones. Toutefois, le codage peut simplifier l'implantation des décodeurs d'état suivant et de sortie. Les deux règles énumérées ci-dessous visent cet objectif.

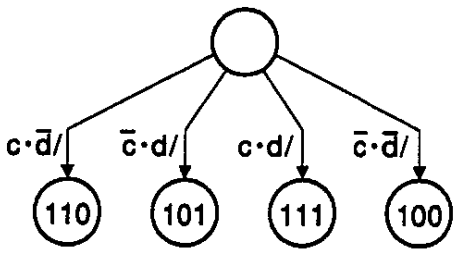
Règle #1 : *Les états ayant le même état suivant pour une même combinaison des commandes externes doivent être codés de telle sorte qu'ils soient en adjacence logique dans la table de Karnaugh.*

Exemple :



Règle #2 : *Les états suivants possibles d'un même état doivent être codés de telle sorte qu'ils soient en adjacence logique dans la table de Karnaugh.*

Exemple :



Note : *En cas de conflit entre les 2 règles de codage, la règle #1 a préséance sur la règle #2. L'application de ces règles n'est pas aisée et ne donne pas toujours les meilleurs résultats.*

Remarques :

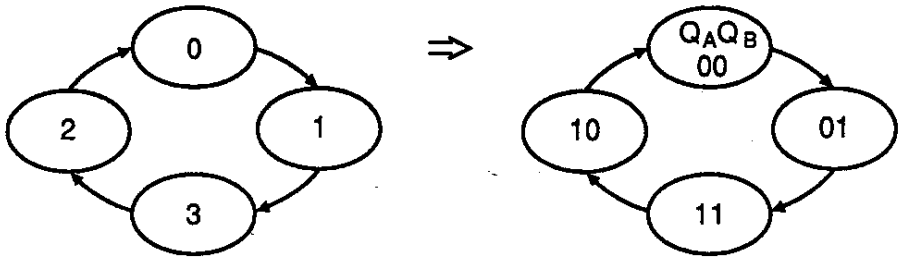
1) La synthèse d'un compteur nécessite les mêmes étapes moins celles faisant allusion au choix du codage et au décodeur de sortie.

2) Le choix du code des états ne pose aucune restriction (contrairement aux circuits séquentiels asynchrones) mais, bien sûr, chaque code doit être relié à un et un seul état. Même si le codage influence le coût, une méthode rigoureuse de la recherche du code n'en vaut pas la peine d'autant plus qu'elle ne s'adresse qu'à une classe restreinte de cas en règle générale.

7.5 - Exemples de synthèse classique

7.5.1 - Premier exemple: Compteur Gray (ou de Johnson) à 2 bits

On veut réaliser un compteur 2 bits à 4 états qui ne change que d'un bit à la fois (compteur Gray). Le compteur ne possède aucune commande externe de sorte que seules les sorties des éléments "mémoire" deviennent les variables indépendantes du décodeur d'état suivant. Comme il s'agit d'un compteur, le décodeur de sortie est absent.



Le graphe ci-dessus est le graphe final : aucunde simplification ne doit et ne peut être faite.

Remarque : Il faut faire les tables pour les bascules D et J-K pour trouver le type de bascule minimisant le coût.

Table état présent/suivant (bascule D)

rappel $Q_n \rightarrow Q_{n+1}$		D
0	0	0
0	1	1
1	0	0
1	1	1

état présent		état suivant		D_A	D_B
Q_{An}	Q_{Bn}	Q_{An+1}	Q_{Bn+1}		
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	1	0

TRANSITION → IMPLIQUE →

Table état présent/suivant (bascule J-K)

rappel $Q_n \rightarrow Q_{n+1}$		J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

état présent Q_{An}	Q_{Bn}	état suivant Q_{An+1}	Q_{Bn+1}	J_A	K_A	J_B	K_B
0	0	0	1	0	X	1	X
0	1	1	1	1	X	X	0
1	0	0	0	X	1	0	X
1	1	1	0	X	0	X	1

TRANSITION — IMPLIQUE

Table de Karnaugh des commandes des éléments "mémoire"

D_A	Q_{An}	
	0	0
Q_{Bn}	1	1

$D_A = Q_{Bn}$

D_B	Q_{An}	
	1	0
Q_{Bn}	1	0

$D_B = \overline{Q_{An}}$

J_A	Q_{An}	
	0	X
Q_{Bn}	1	X

$J_A = Q_{Bn}$

K_A	Q_{An}	
	X	1
Q_{Bn}	X	0

$K_A = \overline{Q_{Bn}}$

J_B	Q_{An}	
	1	0
Q_{Bn}	X	X

$J_B = \overline{Q_{An}}$

K_B	Q_{An}	
	X	X
Q_{Bn}	0	1

$K_B = Q_{An}$

Pour bien comprendre le rôle de ces tables disons simplement qu'elles découlent directement des tables état présent/suivant dressées précédemment et qu'elles définissent le circuit combinatoire placé à chacune des commandes des éléments "mémoire" pour préparer la prochaine transition dépendante de l'état présent $Q_{An}\ Q_{Bn}$.

* lorsque $\begin{cases} Q_{An}=0 \text{ (état 0)} \\ Q_{Bn}=0 \end{cases}$ on veut obtenir $\begin{cases} Q_{An+1}=0 \text{ (état 1)} \\ Q_{Bn+1}=1 \end{cases}$

(Q_{An+1} et Q_{Bn+1} représentent la sortie des bascules au prochain coup d'horloge)

Donc $\begin{cases} D_A = 0 \\ D_B = 1 \end{cases}$ ou $\begin{cases} J_A = 0\ ,\ K_A = X \\ J_B = 1\ ,\ K_B = X \end{cases}$ dans l'état 0

* lorsque $\begin{cases} Q_{An}=0 \\ Q_{Bn}=1 \end{cases}$ (état 1), on veut obtenir $\begin{cases} Q_{An+1}=1 \\ Q_{Bn+1}=1 \end{cases}$ (état 3)

Donc $\begin{cases} D_A = 1 \\ D_B = 1 \end{cases}$ ou $\begin{cases} J_A = 1 \ , \ K_A = X \\ J_B = X \ , \ K_B = 0 \end{cases}$ dans l'état 1

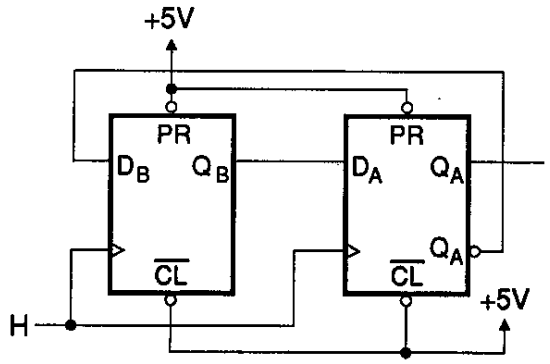
* lorsque $\begin{cases} Q_{An}=1 \\ Q_{Bn}=0 \end{cases}$ (état 2), on veut obtenir $\begin{cases} Q_{An+1}=0 \\ Q_{Bn+1}=0 \end{cases}$ (état 0)

Donc $\begin{cases} D_A = 0 \\ D_B = 0 \end{cases}$ ou $\begin{cases} J_A = X \ , \ K_A = 1 \\ J_B = 0 \ , \ K_B = X \end{cases}$ dans l'état 2

* lorsque $\begin{cases} Q_{An}=1 \\ Q_{Bn}=1 \end{cases}$ (état 3), on veut obtenir $\begin{cases} Q_{An+1}=1 \\ Q_{Bn+1}=0 \end{cases}$ (état 2)

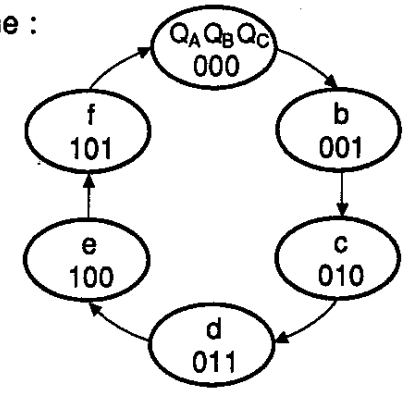
Donc $\begin{cases} D_A = 1 \\ D_B = 0 \end{cases}$ ou $\begin{cases} J_A = X \ , \ K_A = 0 \\ J_B = X \ , \ K_B = 1 \end{cases}$ dans l'état 3

Schéma du circuit avec bascules D



7.5.2 - Compteur binaire modulo 6.

graphe :



On peut identifier le graphe des transitions sur une table de Karnaugh fonction des bits de codage seulement (puisqu'aucune commande externe). Cette table peut donc, à la rigueur, remplacer un graphe avec toute l'information, ce qui n'est pas toujours le cas avec un diagramme de fonctionnement lorsque le système se complique.

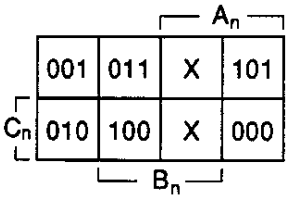


Table état présent/suivant (bascules D)

état présent			état suivant			D_A	D_B	D_C
Q_{An}	Q_{Bn}	Q_{Cn}	Q_{An+1}	Q_{Bn+1}	Q_{Cn+1}			
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

→ états impossibles que le système ne peut prendre lors d'un fonctionnement normal avec l'initialisation asynchrone dans un état permis.

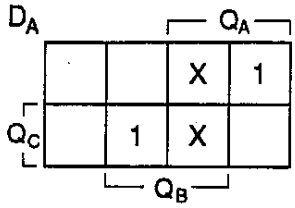
* *Pour un compteur binaire, de part sa structure intrinsèque de comptage, on comprend qu'il finit toujours par s'autocorriger. La notion d'autocorrection est surtout utile pour les compteurs en anneau et les compteurs de Johnson.*

Table état présent/suivant (bascule J-K)

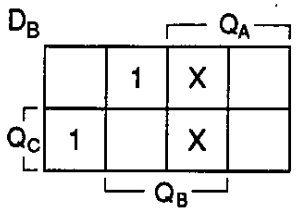
état présent			état suivant			J_A	K_A	J_B	K_B	J_C	K_C
Q_{An}	Q_{Bn}	Q_{Cn}	Q_{An+1}	Q_{Bn+1}	Q_{Cn+1}						
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

Note : À l'avenir, les indices n et n+1 attachés aux bits de codage tomberont sans semer de confusion. Il faut toutefois se souvenir de la relation entre Q_{in} et $Q_{i\,n+1}$.

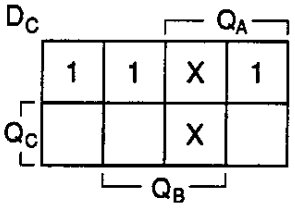
Tables de Karnaugh des commandes des éléments "mémoire"



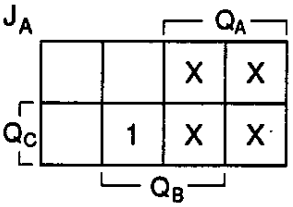
$D_A = Q_A\bar{Q}_C + Q_BQ_C$



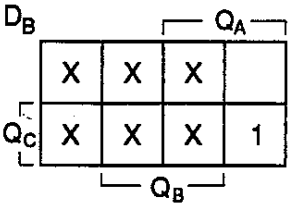
$D_B = Q_B\bar{Q}_C + \bar{Q}_A\bar{Q}_BQ_C$



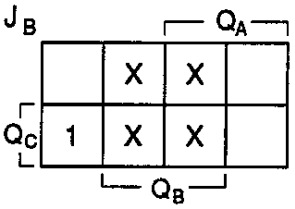
$D_C = \bar{Q}_C$



$J_A = Q_BQ_C$

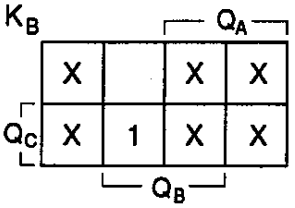


$K_A = Q_C$



$J_B = \bar{Q}_AQ_C$

$J_C = 1$

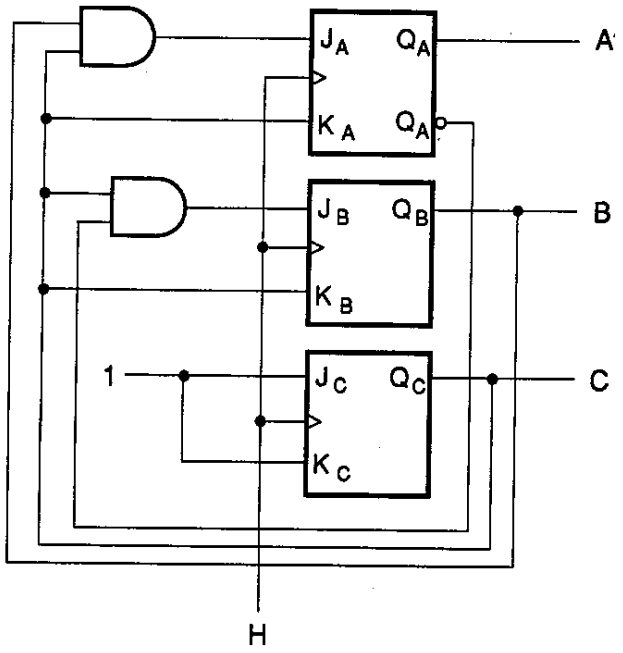


$K_B = Q_C$

$K_C = 1$

On remarque qu'ici encore, les tables de Karnaugh des commandes J et K contiennent des valeurs indifférentes dans au moins la moitié des états. Ceci peut résulter en des bénéfices lors de la simplification. Cette particularité est générale et découle de la table d'excitation de la bascule J-K.

Réalisation avec bascules J-K



7.5.3 - Graphe avec commandes externes.

Jusqu'à présent, les graphes des circuits synchrones à réaliser étaient exempts de commandes externes. Évidemment, la grande majorité des circuits séquentiels suivent des séquences non-répétitives dictées par des signaux externes qui agissent comme commandes. Le "monde extérieur" peut donc modifier la séquence du circuit synchrone (i.e. le chemin suivi sur le graphe des états).

La synthèse des circuits séquentiels synchrones avec commandes externes suit les étapes régulières. La seule différence avec les 2 exemples précédents consiste à traiter les variables de commande comme les bits de codage. Ainsi, les variables de commande et les bits de codage agissent comme variables indépendantes dans la synthèse des décodeurs d'état suivant et de sortie. La table état présent/ suivant (ou la table de transition applicable dans ces cas) se construit en tenant compte des commandes externes pour effectuer une des transitions possibles à partir d'un état présent.

Soit le graphe ci-dessous montrant 2 commandes externes (CF et CG) où les bits de codage sont les sorties du système. Le tout ressemble à un compteur binaire 2 bits avec commandes d'attente dans 2 états.

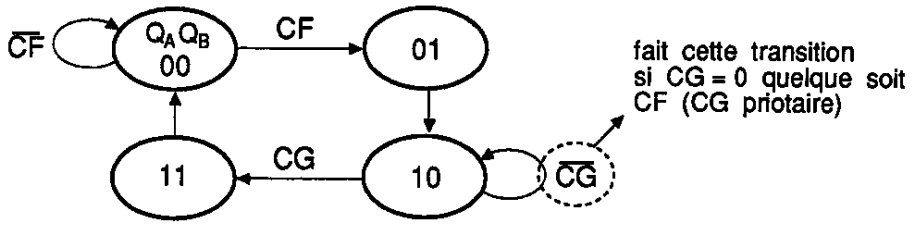


Table état présent/suivant

état présent Q_A	Q_B	commandes CF	CG	état suivant Q_A	Q_B	J_A	K_A	J_B	K_B	D_A	D_B
0	0	0	X	0	0	0	X	0	X	0	0
		1	X	0	1	0	X	1	X	0	1
0	1	X	X	1	0	1	X	X	1	1	0
1	0	X	0	1	0	X	0	0	X	1	0
		X	1	1	1	X	0	1	X	1	1
1	1	X	X	0	0	X	1	X	1	0	0

transition de l'état Q_AQ_B=10
à l'état Q_AQ_B=10 lorsque
CG=0 quel que soit CF.

Théoriquement, la table état présent/suivant devrait comporter 16 combinaisons (2^4, 2 bits de codage + 2 bits de commande) mais l'inaction des commandes dans certains états nous permet de faire quelques simplifications.

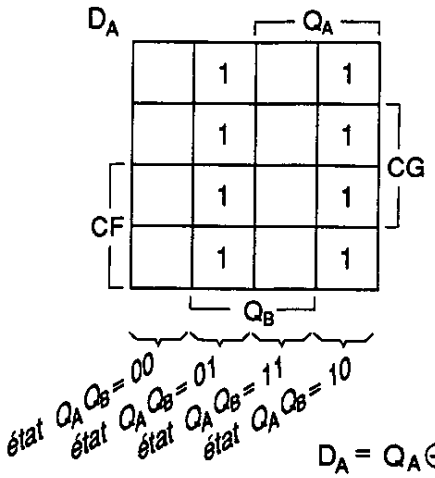
Table de transition

La table de transition conserve la même information que celle contenue dans un graphe. Elle indique $Q_{An+1}\,Q_{Bn+1}$ en fonction de Q_{An}, Q_{Bn}, CF_n et CG_n. L'avantage de la table de transition est sa construction rapide et plus condensée que la table état présent/suivant lorsqu'il y a des commandes externes. Elle présente ce cependant un inconvénient pour l'ingénieur inexpérimenté pour la synthèse avec bascules J-K.

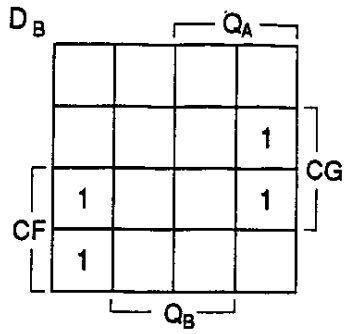
état présent Q_AQ_B	CF,CG 00	01	10	11
00	00	00	01	01
01	10	10	10	10
10	10	11	10	11
11	00	00	00	00

transition de l'état Q_AQ_B=10
à l'état Q_AQ_B=10 lorsque
CG=0 (CF CG=00, CF CG=10)
ou à l'état Q_AQ_B=11 lorsque
CG=1 (CF CG=01, CF CG=11)

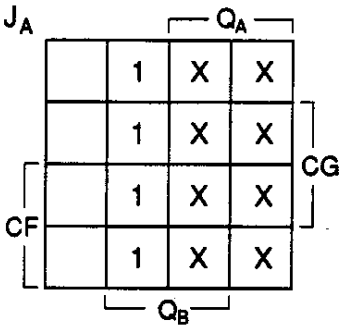
Tables de Karnaugh des commandes des éléments "mémoire"



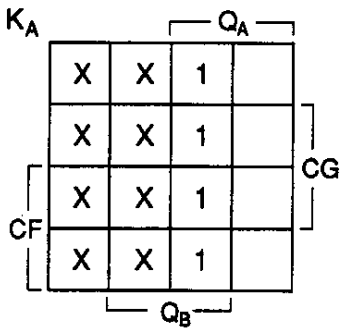
$D_A = Q_A \oplus Q_B$



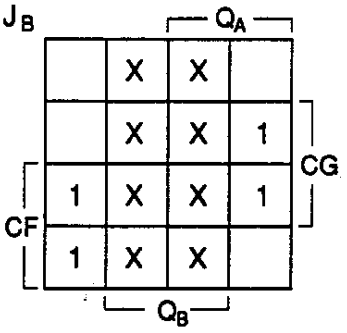
$D_B = \overline{Q}_A \overline{Q}_B CF + Q_A \overline{Q}_B CG$



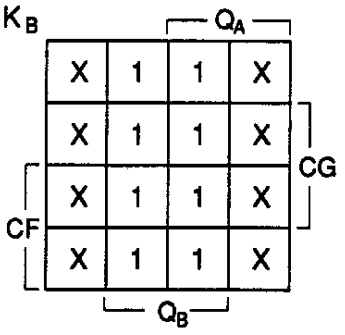
$J_A = Q_B$



$K_A = Q_B$

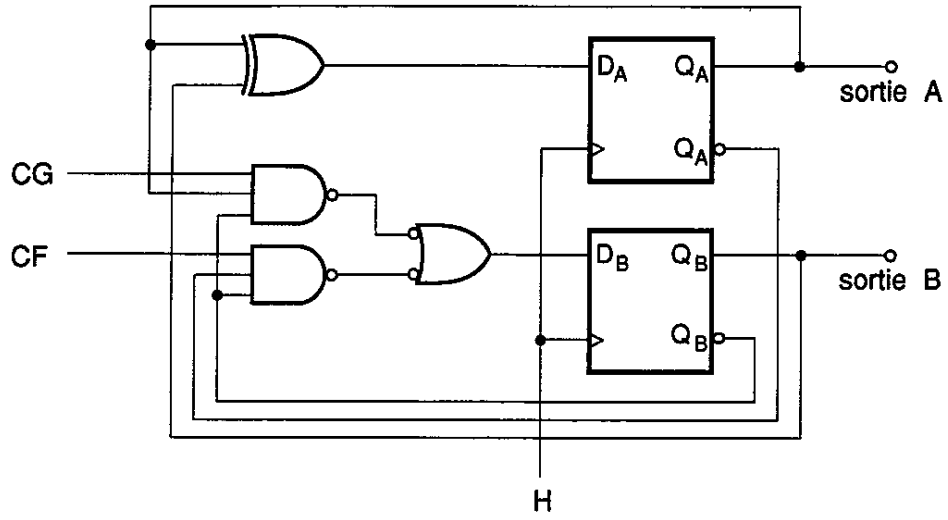


$J_B = Q_A \cdot CG + \overline{Q}_A \cdot CF$



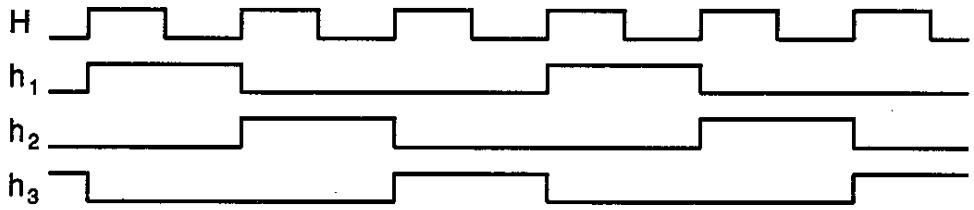
$K_B = 1$

Réalisation avec bascules D



7.5.4 - Commandes étatiques (horloge 3 phases)

Une phase est un blip qui est déphasé par rapport aux autres phases crées par le circuit. Ainsi, une horloge 3 phases possède 3 sorties (chacune représentant une phase) actives pendant une seule période d'horloge, à toutes les 3 périodes comme l'illustre le diagramme de fonctionnement tracé ci-dessous.



On identifie un signal agissant comme phase par le symbole (h_i). Attention, les phases ne sont pas des commandes de synchronisme comme l'horloge mère dans leur but premier. Elles servent plutôt dans des systèmes synchrones complets comprenant des éléments fonctionnels commandés par un séquenceur (cf. chapître 8): les phases produites par le séquenceur dirigent les opérations dictées par un organigramme sur les commandes synchrones des éléments fonctionnels.

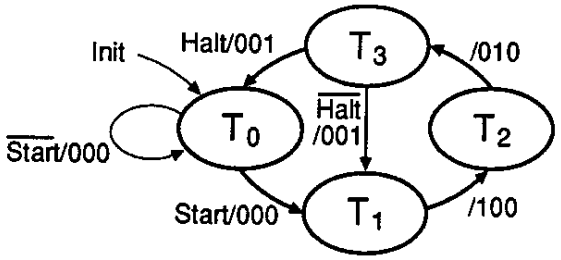
Cet exemple, développé de 2 façons, illustre 3 nouvelles situations dans la synthèse de circuits séquentiels synchrones.

- les commandes externes étatiques
- le jeu du codage des états
- l'introduction de variables conditionnelles

Le circuit possède 2 commandes externes :

* **Start :** blip de démarrage qui n'arrive que lorsque le circuit est dans l'état de repos T_0
* **Halt :** Signal d'arrêt qui est susceptible de se produire dans tous les états du circuit mais qui n'est effectif que si le circuit est dans l'état T_3

Une mise à zéro (Init) asynchrone amène le circuit dans l'état de repos T_0.



format de branchement
$/h_1\ h_2\ h_3$

On dit, ici, que la commande externe "Start" est étatique puisqu'elle n'est pas active que dans certains états (T_0).

a) Codage sur 2 bits

état	Q_AQ_B
T_0	0 0
T_1	0 1
T_2	1 0
T_3	1 1

Le décodeur de sortie est donc essentiel car les bits de codage ne correspondent pas aux sorties de l'horloge 3 phases.

Table état Présent/suivant

état présent Q_AQ_B	Start	Halt	état suivant Q_AQ_B	J_AK_A	J_BK_B	D_AD_B
0 0	0	X	0 0	0 X	0 X	0 0
	1	X	0 1	0 X	1 X	0 1
0 1	0	X	1 0	1 X	X 1	1 0
	1	X	X X	X X	X X	X X
1 0	0	X	1 1	X 0	1 X	1 1
	1	X	X X	X X	X X	X X
1 1	0	0	0 1	X 1	X 0	0 1
	0	1	0 0	X 1	X 1	0 0
	1	X	X X	X X	X X	X X

"Start" ne peut être actif dans ces états donc l'état suivant est indifférent

Table de transition

(lorsqu'on maîtrise bien les tables caractéristiques des éléments "mémoire")

état présent Q_AQ_B	Start, Halt 0 0	0 1	1 0	1 1
0 0	0 0	0 0	0 1	0 1
0 1	1 0	1 0	X	X
1 0	1 1	1 1	X	X
1 1	0 1	0 0	X	X

Tables de Karnaugh des commandes des éléments "mémoire"

D_A

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0		1		1
0 1		1		1
1 1		X	X	X
1 0		X	X	X

Q_B

$D_A = Q_A \oplus Q_B$

D_B

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0			1	1
0 1				1
1 1	1	X	X	X
1 0	1	X	X	X

Q_B

$D_B = Start + Q_A\overline{Q}_B + \overline{Halt} \cdot Q_A$

J_A

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0		1	X	X
0 1		1	X	X
1 1		X	X	X
1 0		X	X	X

Q_B

$J_A = Q_B$

K_A

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0	X	X	1	
0 1	X	X	1	
1 1	X	X	X	X
1 0	X	X	X	X

Q_B

$K_A = Q_B$

J_B

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0		X	X	1
0 1		X	X	1
1 1	1	X	X	X
1 0	1	X	X	X

Q_B

$J_B = Q_A + Start$

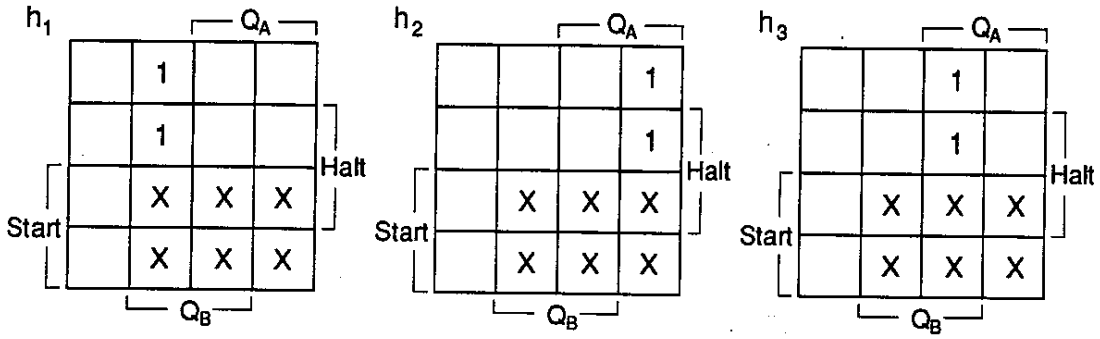
K_B

Start \ Halt	Q_A 0 0	0 1	1 1	1 0
0 0	X	1		X
0 1	X	1	1	X
1 1	X	X	X	X
1 0	X	X	X	X

Q_B

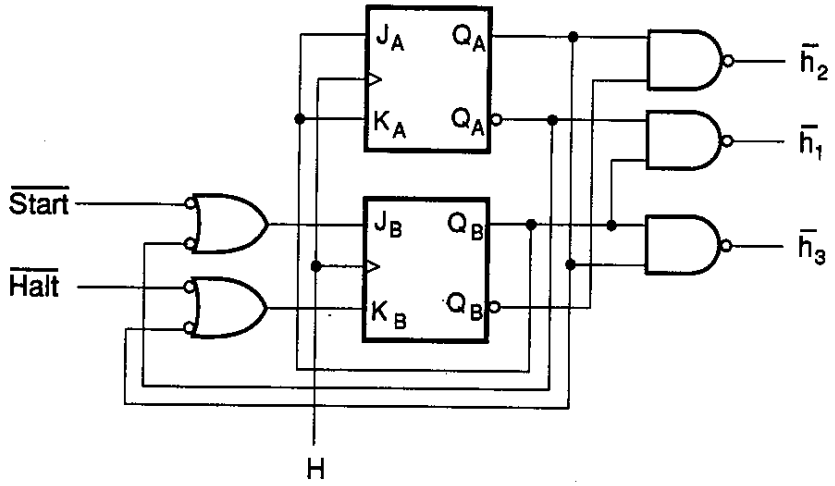
$K_B = \overline{Q}_A + Halt$

Décodeur de sortie



$h_1 = \overline{Q}_A \cdot Q_B$ $\quad h_2 = Q_A \cdot \overline{Q}_B$ $\quad h_3 = Q_A Q_B$

Réalisation avec bascules J-K



b) Codage sur 3 bits en anneau

état	$Q_A Q_B Q_C$
T_0	0 0 0
T_1	1 0 0
T_2	0 1 0
T_3	0 0 1

De cette manière, on élimine le décodeur de sortie: la phase h_1 pouvant être Q_A ; $h_2 = Q_B$ et $h_3 = Q_C$. L'avantage semble évident mais il y a toutefois un inconvénient : les tables de Karnaugh comportent 5 v.i (3 bits de codage et 2 bits de commande).

Pour outre-passer la difficulté, on peut traiter les commandes "Halt" et "Start" comme des variables conditionnelles (VEM). La diminution du travail avec VEM est d'autant plus grande lorsque quelques-unes ou plus des variables conditionnelles sont étatiques.

En employant seulement les bits de codage comme variables indépendantes des sub-minterms (i.e. toutes les commandes externes agissent comme VEM), alors chaque case de la table de Karnaugh représente un état possible de la machine. Cette façon de procéder simplifie l'usage des VEM lorsque certaines variables sont étatiques (cf. paragraphe 3.11.3). Le mieux est d'indiquer dans un des coins de chaque case, les commandes externes possiblement actives dans l'état correspondant et sous-entendre que toutes les externes non-actives surement sont multipliées par X puis sommées. La simplification est d'autant plus efficace.

tableau de transition

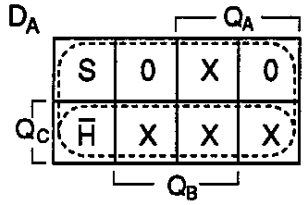
état présent $Q_AQ_BQ_C$	Start, Halt 0 0	0 1	1 0	1 1
0 0 0	0 0 0	0 0 0	1 0 0	1 0 0
0 0 1	1 0 0	0 0 0	X	X
0 1 0	0 0 1	0 0 1	X	X
0 1 1	X	X	X	X
1 0 0	0 1 0	0 1 0	X	X
états impossibles 1 0 1	X	X	X	X
états impossibles 1 1 0	X	X	X	X
états impossibles 1 1 1	X	X	X	X

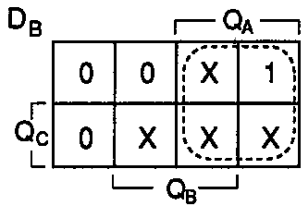
Comme le code ressemble à la séquence produite par un compteur en anneau, il est logique de penser que la réalisation avec bascules D sera plus simple (voir p.4-38).

Les tables de Karnaugh des commandes D_i sont les suivantes :

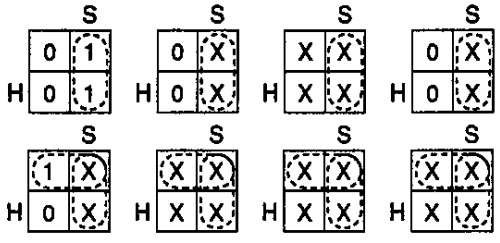
NOTATION : Start = 'S'
Halt = 'H'

TABLES DE KARNAUGH

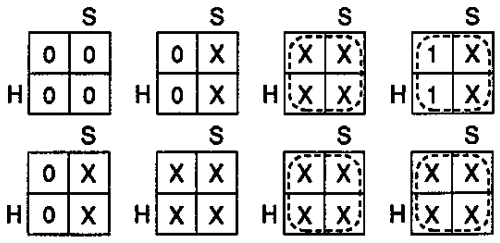




TABLES DE LAURENDEAU !

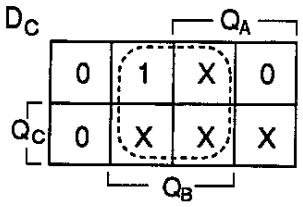


$D_A = S + \bar{H} \cdot Q_C$

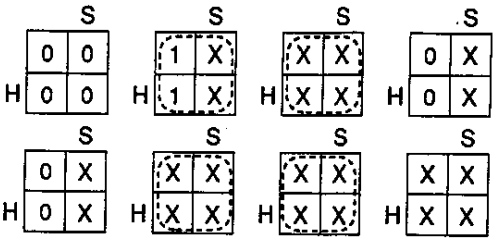


$D_B = Q_A$

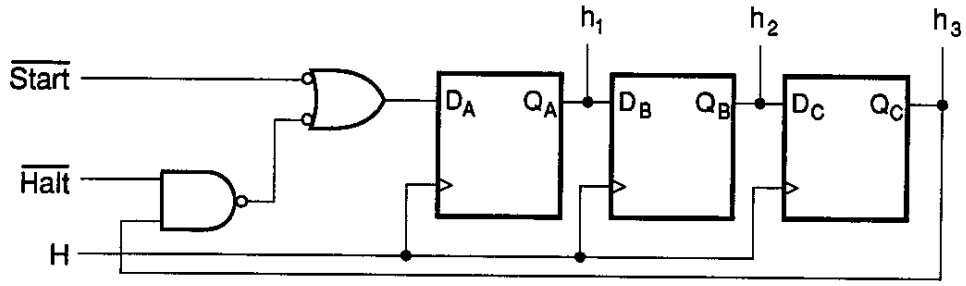
TABLES DE KARNAUGH (suite)



TABLES DE LAURENDEAU ! (suite)



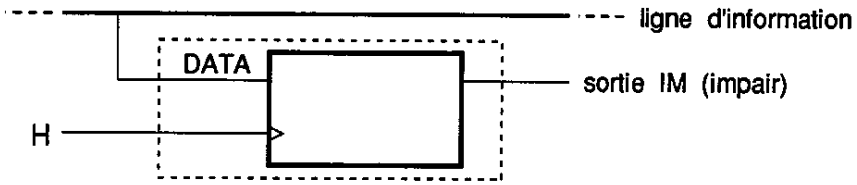
Réalisation



7.5.5 - Réduction des états et codage.

Une ligne d'information série est échantillonnée 3 fois (un échantillon à chaque montée de l'horloge). Si et seulement si 2 échantillons sont actifs haut, on procède à un 4ème échantillonnage. La sortie du système indique un nombre impair d'échantillons actifs haut à la fin du cycle marqué par un retour à l'état initial pour un nouveau cycle.

Les échantillons peuvent être vus comme la commande externe du système comme le démontre la figure ci-dessous.



Construction du graphe primitif

La valeur de l'échantillon à chaque échantillonnage donne :

2 transitions possibles → DATA
→ $\overline{DATA}$

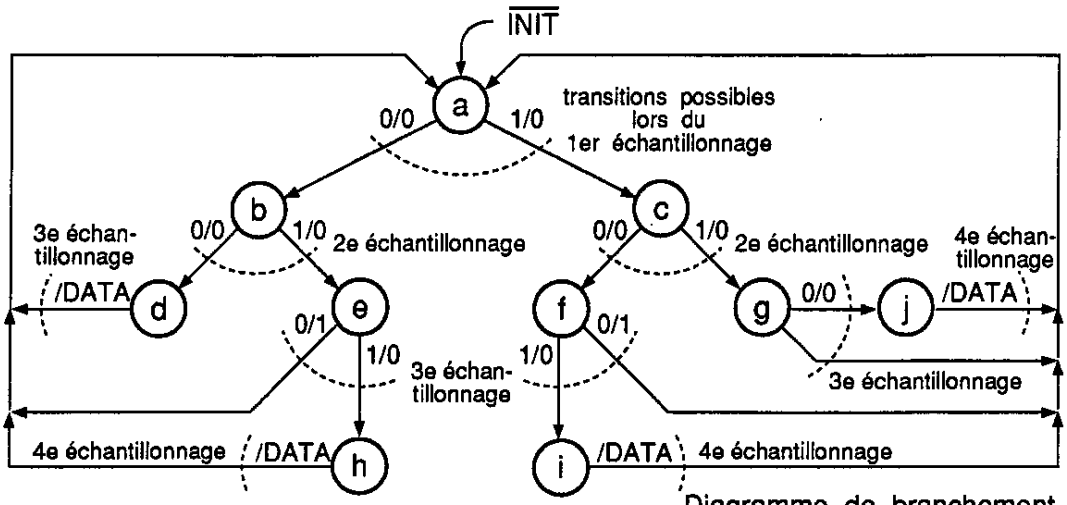


Diagramme de branchement DATA /IM

Table état présent/suivant primitive

état présent	DATA	état suivant	IM
a	0	b	0
	1	c	0
b	0	d	0
	1	e	0
c	0	f	0
	1	g	0
d	0	a	0
	1	a	1
e	0	a	1
	1	h	0
f	0	a	1
	1	i	0
g	0	j	0
	1	a	1
h	0	a	0
	1	a	1
i	0	a	0
	1	a	1
j	0	a	0
	1	a	1

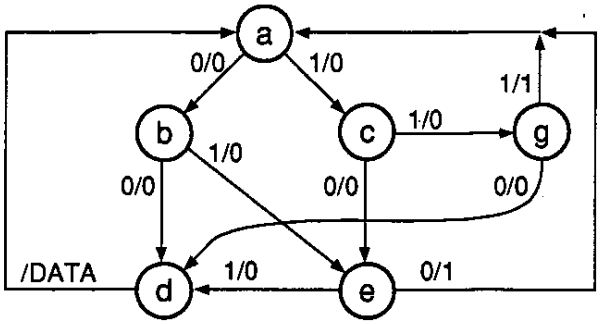
Puisque h EQ i
e EQ f

d EQ h, d EQ i, d EQ j

Après simplifications

a	0	b	0
	1	c	0
b	0	d	0
	1	e	0
c	0	e	0
	1	g	0
d	0	a	0
	1	a	1
e	0	a	1
	1	d	0
g	0	d	0
	1	a	1

Graphe simplifié

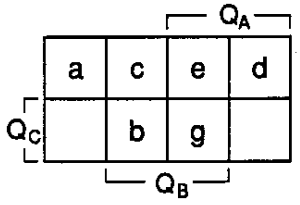


Codage

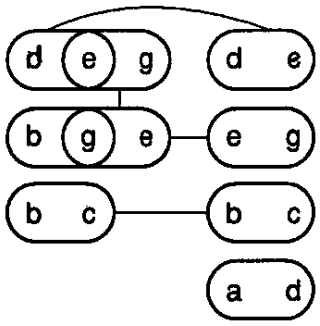
état(s) précédent(s)	état présent	état(s) suivant(s)
d,e,g	a	b,c
a	b	d,e
a	c	e,g
b,g,e	d	a
b,c	e	a,d
c	g	a,d
règle #1 appliquable		règle #2 appliquable

⇒ **règle #1** Les états "d","e" et "d","g" doivent être adjacents pour qu'ils puissent être regroupés dans la table de Karnaugh. Même chose pour les états "b" et "g".

⇒ **règle #2** Les états "b" et "c" doivent être adjacents
"d" et "e"
"e" et "g"
"a" et "d"



Position des états dans la table de Karnaugh ou les v.i. sont les bits de codage.



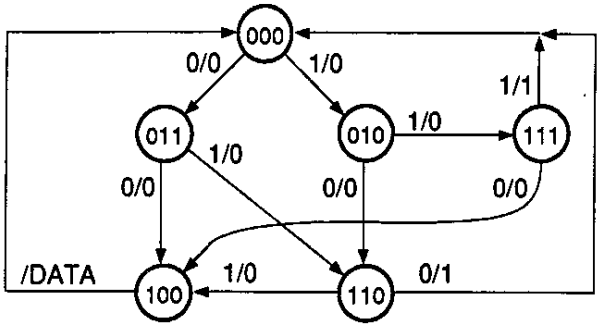
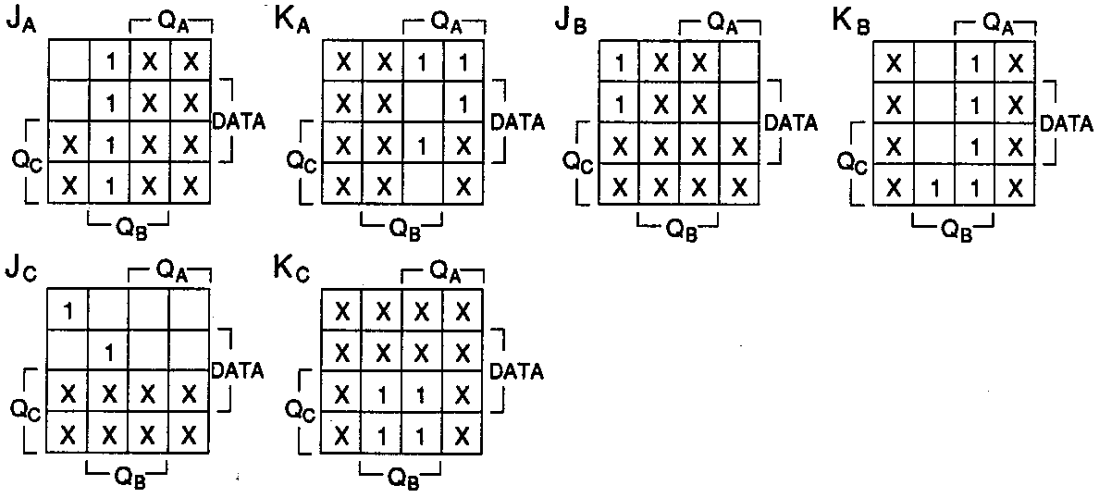


Table état présent/suivant simplifiée

état présent	DATA	état suivant	IM
000	0	011	0
	1	010	0
010	0	110	0
	1	111	0
011	0	100	0
	1	110	0
100	0	000	0
	1	000	1
110	0	000	1
	1	100	0
111	0	100	0
	1	000	1

Table de Karnaugh des commandes J et K



$J_A = Q_B$

$K_A = Q_C \bullet DATA + \overline{Q}_C \bullet \overline{DATA} + \overline{Q}_B = \overline{Q_C \oplus DATA} + \overline{Q}_B$

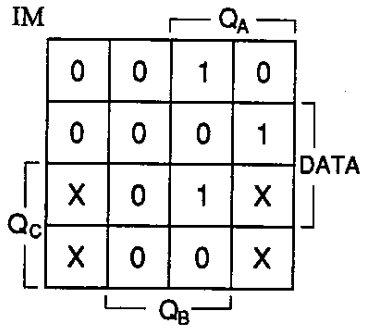
$J_B = \overline{Q}_A$

$K_B = Q_C \bullet \overline{DATA} + Q_A$

$J_C = \overline{Q}_A \overline{Q}_B \bullet \overline{DATA} + \overline{Q}_A \bullet Q_B \bullet DATA = \overline{Q}_A (\overline{Q_B \oplus DATA})$

$K_C = 1$

Décodeur de sortie



$$IM = Q_A Q_B \bar{Q}_C \cdot \overline{DATA} + Q_A \bar{Q}_B DATA + Q_A Q_C DATA$$

note: avec bascules D on trouve

$$D_A = Q_B \cdot \bar{Q}_C \cdot DATA + Q_C \cdot \overline{DATA} + Q_A \bar{Q}_B$$

$$D_B = \bar{Q}_A \cdot DATA + \bar{Q}_A \bar{Q}_C$$

$$D_C = \bar{Q}_A Q_B \bar{Q}_C \cdot DATA + \bar{Q}_A \bar{Q}_B \cdot \overline{DATA}$$

7.5.6 - Système de transmission série

Cet exemple montre la complexité de la synthèse classique d'un circuit séquentiel synchrone répondant à un graphe faisant intervenir plusieurs commandes externes. L'exemple sera repris à maintes occasions pour illustrer les transformations possibles de la synthèse. Ces transformations simplifieront le travail de synthèse du décodeur d'état suivant notamment, car cette partie prend des proportions énormes lorsqu'il y a plusieurs commandes. Elles utilisent alors les circuits intégrés à plus haut niveau d'intégration que celui des bascules simples.

Le système de transmission série proposé ici regroupe les étapes simples, de base des systèmes de transmission série disponibles aujourd'hui. Il n'est pas nécessaire de comprendre à fond le principe du système ; il s'agissait ici de définir un graphe sous prétexte d'être réaliste.

But : *Faire la synthèse d'un circuit suivant la séquence nécessaire pour lire un bloc de données enregistré sur un ruban magnétique et de transmettre ce bloc en série (un bit derrière l'autre sur un seul fil) en ajoutant des bits de codage (code correcteur d'erreur).*

Principe : *Le système, à l'initialisation, entre dans un état de repos, attendant un signal de lecture. Lorsque ce dernier arrive, il lit un bloc de données sur le ruban et emmagasine ce bloc dans une mémoire tampon ("buffer"). Il commence alors la transmission série de ce qui est dans la mémoire tampon en ajoutant des bits de codage à la fin de chaque mot. Entre chaque mot, il attend un avis de réception indiquant s'il faut retransmettre le dernier mot (mauvaise réception: erreurs causées par le bruit dans le canal de transmission).*

Graphe :

codage: $Q_A Q_B Q_C$

états

Repos

Lecture d'un bloc écriture dans la mémoire tampon (sortie LOAD = 1)

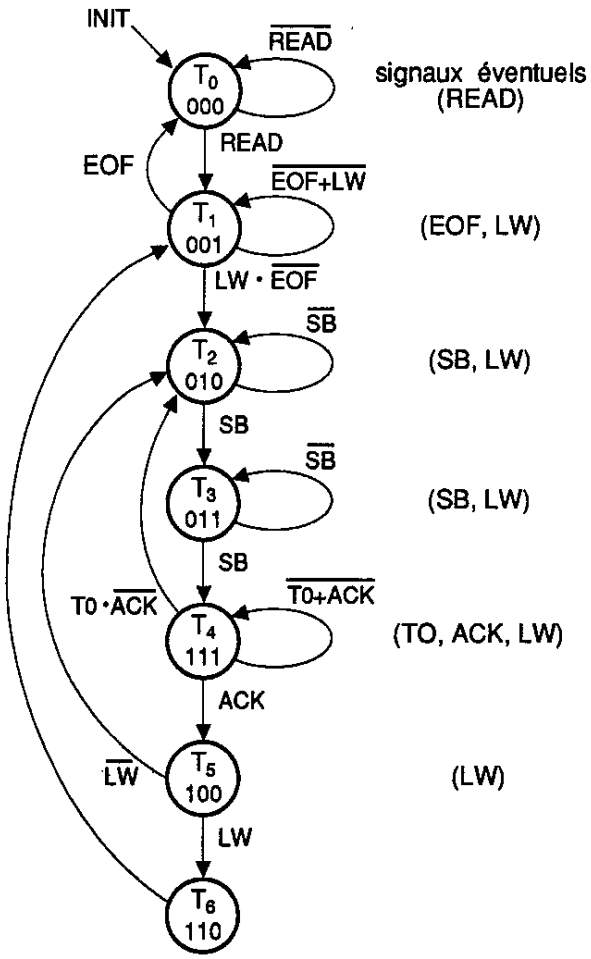
Transmission d'un bit (sortie TRANSMIT = 1)

Transmission d'un mot de codage (sortie TRANSMIT = 1)

Attente de l'avis de réception

Redémarrage (sortie READY = 1)

Préparation pour lecture d'un autre bloc (sortie READY = 1)



signaux	définition	états d'activation possible
READ	blip de démarrage (ne peut survenir que lorsqu'on est en T_0)	T_0
EOF	mot indiquant une fin de fichier ("end of file")	T_1
SB	indique le dernier bit d'un mot ou du code (stop bit")	T_2 , T_3
TO	demande de retransmission ("time out")	T_4
ACK	avis de réception ("acknowledge")	T_4
LW	indique le dernier mot d'un bloc ("last word")	T_1 , T_2 , T_3 , T_4 , T_5

Le nombre de variables indépendantes impliquées dans la synthèse (6 variables d'entrée + 3 variables d'état) donne l'avantage à l'utilisation des VEM. On pourra, par la même occasion, exploiter le phénomène étatique des commandes externes.

Tables de Karnaugh des commandes J_i et K_i

J_A

Q_C \ Q_B	0	1
0	0	X
1	SB	X

$J_A = SB \cdot Q_C$

K_A

Q_C \ Q_B	0	1
0	X	X
1	X	TO·$\overline{ACK}$

$K_A = TO \cdot \overline{ACK} + \overline{LW} \cdot \overline{Q}_C + Q_B \cdot \overline{Q}_C$

(dans l'état 6, $LW \cdot Q_C = 1$ obligatoirement puisque $\overline{LW}=1$)

J_B

Q_C \ Q_B	0	1
0	X	X
1	LW·$\overline{EOF}$	X

$J_B = LW \cdot \overline{EOF} + Q_A$

K_B

Q_C \ Q_B	0	1
0	X	0
1	X	ACK

$K_B = ACK + Q_A \cdot \overline{Q}_C$

J_C

Q_C \ Q_B	0	1
0	READ	SB
1	X	X

$J_C = READ + SB + Q_A \cdot Q_B$

K_C

Q_C \ Q_B	0	1
0	X	X
1	LW+EOF	0

$K_C = EOF + TO + ACK + LW \cdot \overline{Q}_B$

Note : *états / signaux éventuels*

Q_C \ Q_B	0	1
0	T_0	T_2
1	T_1	T_3

Q_C \ Q_B	0	1
0	T_6	T_5
1	T_4	X

Le décodeur d'état suivant se réalise en près de 3 boitiers. Sa réalisation avec des bascules "D" est beaucoup plus difficile.

décodeur de sortie

LOAD = $\overline{Q}_B Q_C$

TRANSMIT = $\overline{Q}_A Q_B$

READY = $Q_A \overline{Q}_C$

7.6 - Synthèse avec MSI et LSI.

L'exemple de la transmission série confirme que le travail exigé par la synthèse classique croît avec complexité du graphe. Et encore, le travail a été simplifié par des commandes étatiques.

Cela étant, il semble pertinent de modifier la synthèse des circuits séquentiels synchrones par l'emploi à bon escient des circuits intégrés à haut niveau d'intégration tels les multiplexeurs, les ROM et même les PLA, lorsque la complexité s'accroît. D'ailleurs, la gamme de plus en plus variée et le coût de plus en plus faible de ces produits favorisent leur utilisation dans tout design.

7.6.1 - Multiplexeurs directement adressés.

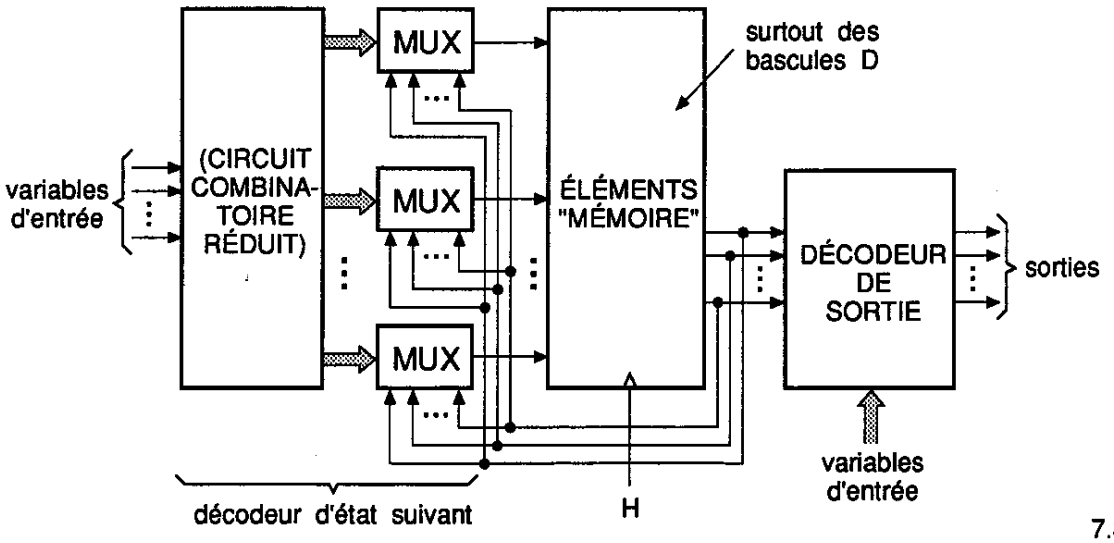
Dans une philosophie d'utilisation des MSI ou LSI, un ensemble de multiplexeurs peut être utilisé dans l'implantation du décodeur d'état suivant. Ces multiplexeurs réduisent la grosseur du circuit combinatoire nécessaire.

L'arrangement des multiplexeurs peut être réalisé comme sur la figure ci-dessous. C'est la configuration avec multiplexeurs directement adressés. L'étude de la configuration révèle que chaque commande des éléments "mémoire" dépend de l'état présent de la machine ainsi que des commandes externes.

commandes des éléments "mémoire" → variables d'état
variables d'entrée

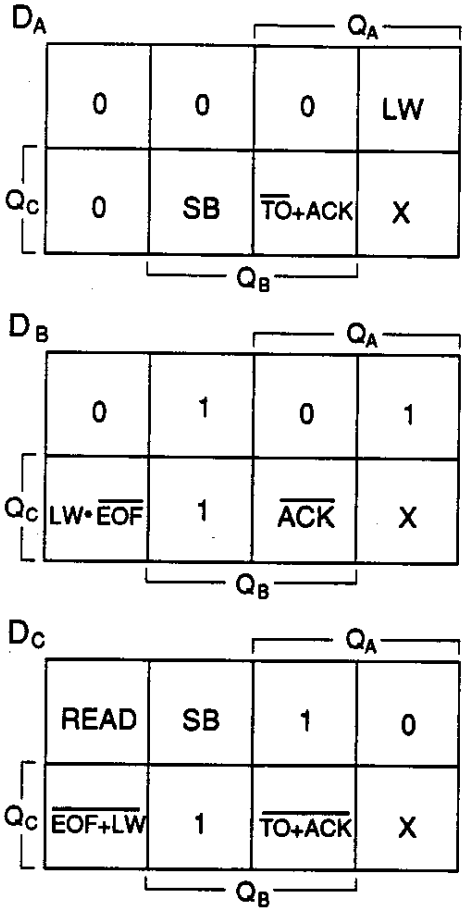
Ainsi, chaque multiplexeur est directement utilisé pour sélectionner la condition de branchement appropriée des variables d'entrée dépendamment des variables d'état.

- **sortie des MUX = commande d'un élément "mémoire"**
- **entrées information des MUX = condition de branchement**
- **entrées adresse des MUX = bits de codage de l'état**



La base de la configuration avec multiplexeurs directement adressés est que les commandes des éléments "mémoire" ne sont fonction que de quelques-unes des variables d'entrée dans un état en particulier. La synthèse demande donc de dresser les tables de Karnaugh des commandes des éléments "mémoire" à l'aide des VEM. Chaque case représente un état de la machine. L'expression à l'intérieur de chaque case ne contient alors que les variables d'entrée et c'est cette expression qui agit à l'entrée information du multiplexeur

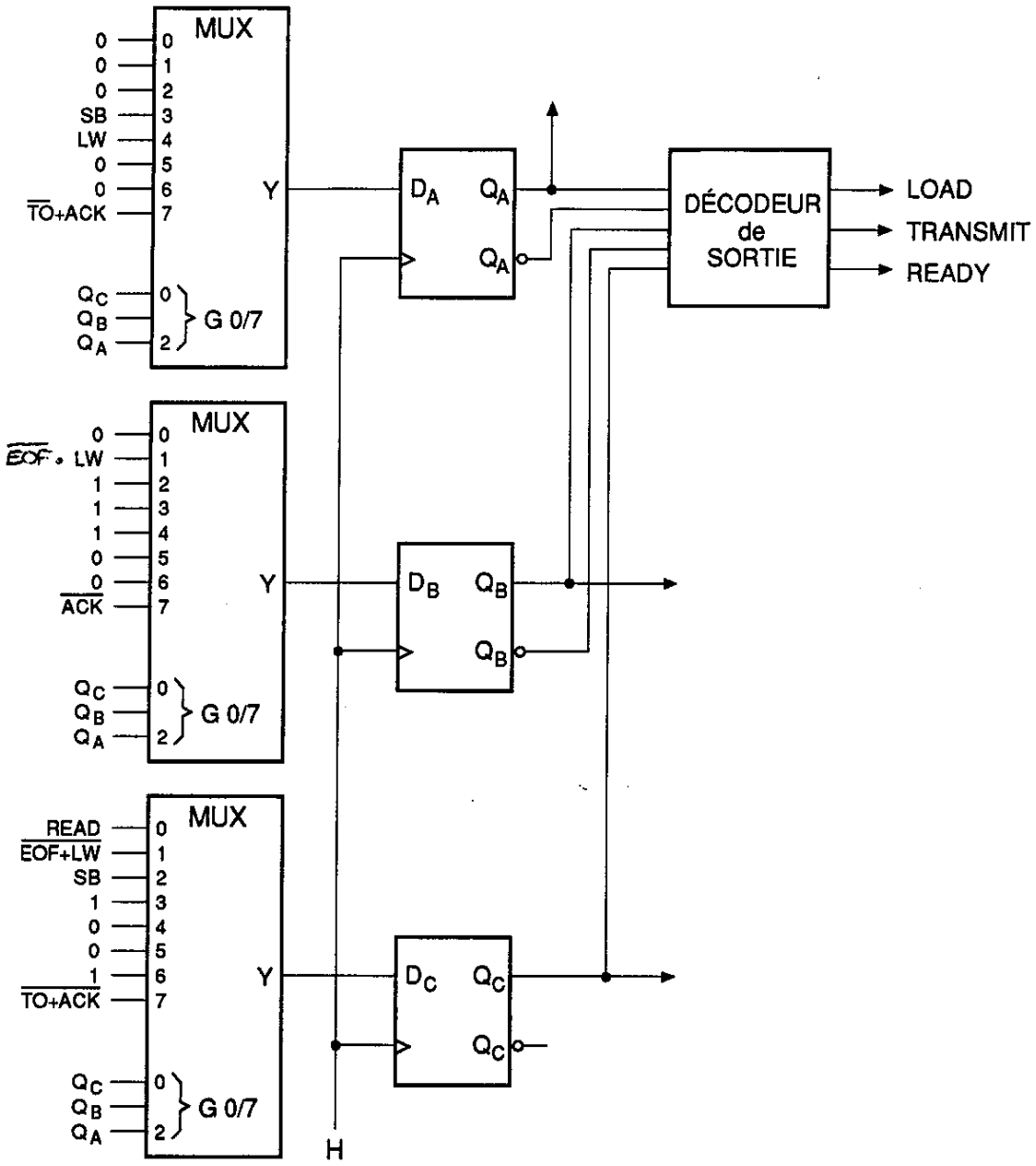
Exemple : Système de transmission série (voir le graphe au paragraphe 7.5.6)



il faut 3 MUX 8 à 1 - un pour l'entrée D_A
- un pour l'entrée D_B
- un pour l'entrée D_C

chaque MUX est adressé par les bits Q_A,Q_B et Q_C ou Q_A à poids fort.

Les 8 entrées information de chaque MUX sont les expressions à l'intérieur de chaque case. Attention, le rang de l'entrée information correspond à la valeur binaire des bits de codage et non au numéro de l'état.



Une modification légère du graphe n'entraine pas un changement important du câblage.

7.6.2 - Multiplexeurs indirectement adressés.

La configuration avec multiplexeurs indirectement adressés diffère de celle vue précedemment par l'ajout d'un circuit combinatoire (avec SSI, ROM ou PLA selon la taille du circuit séquentiel) qui effectue une transformation entre les bits de décodage et l'adresse présentée à un multiplexeur en particulier. C'est le décodeur d'adresse des MUX.

décodeur d'adresse des MUX

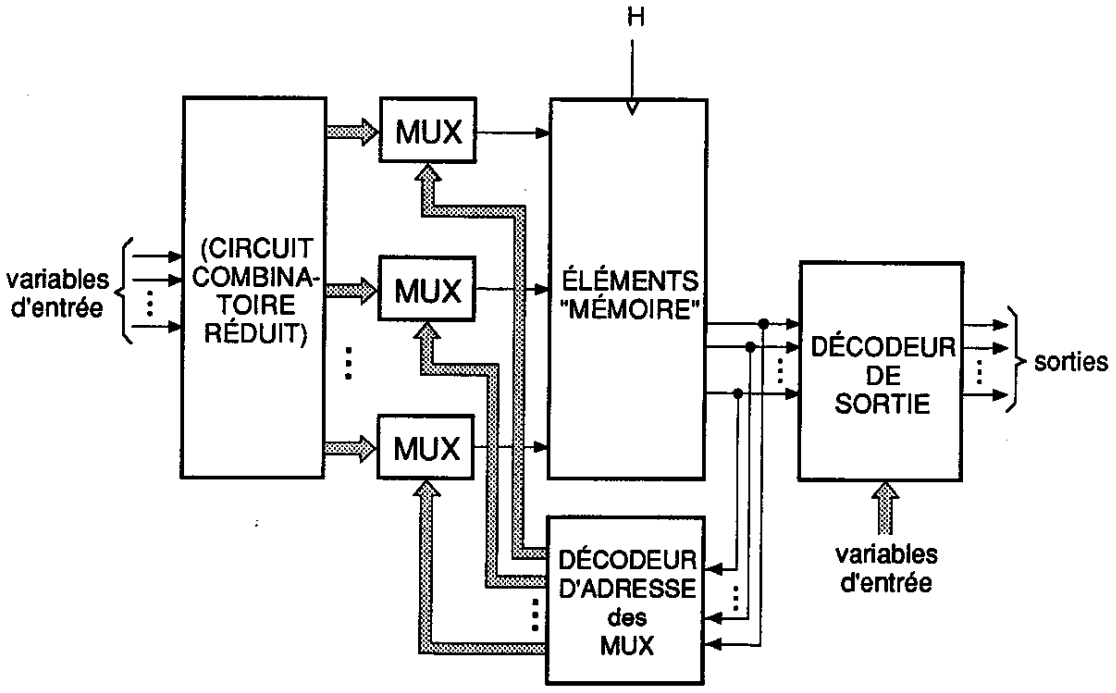
- convertit les bits de codage en bits d'adresse des MUX
- circuit combinatoire → SSI
  ROM ou PLA

Le but du décodeur d'adresse des MUX est de réduire la dimension des MUX employés dans le décodeur d'état suivant. En effet, dans la configuration avec MUX directement adressés, il arrive souvent qu'une expression se répète a plusieurs entrées information d'un MUX, il convient donc de sélectionner l'une des expressions possibles sur un MUX dépendamment de l'état présent.

→ **sortie des MUX =** commande d'un élément "mémoire"
→ **entrées information des MUX =** condition de branchement
→ **entrées adresse des MUX =** sorties du décodeur d'adresse des MUX
→ **entrées du décodeur d'adresse des MUX =** bits de codage de l'état

La technique de synthèse selon cette configuration ressemble à celle utilisée en multiplexeurs directement adressés: on dresse d'abord des tables de Karnaugh des commandes des éléments "mémoire" à l'aide des VEM (les VEM = les variables d'entrée). Il s'agit alors d'assigner une entrée information à chaque expression rencontrée dans la table. Le décodeur d'adresse des MUX est synthétisé de sorte que l'on sélectionne la bonne adresse (donc la bonne expression de branchement) selon l'état présent.

C'est le nombre d'adresses différentes qui détermine la dimension du multiplexeur.



Exemple : Système de transmission série (voir le graphe au paragraphe 7.5.6)

On avait obtenu (cf. exemple du paragraphe 7.6.1)

D_A

Q_C \ Q_B	0	0	1	1
0	0	SB	$\overline{TO}$+ACK	X
0	0	0	LW	

D_B

Q_C \ Q_B	0	1	1	0
LW•$\overline{EOF}$	1	$\overline{ACK}$	X	
0	1	0	1	

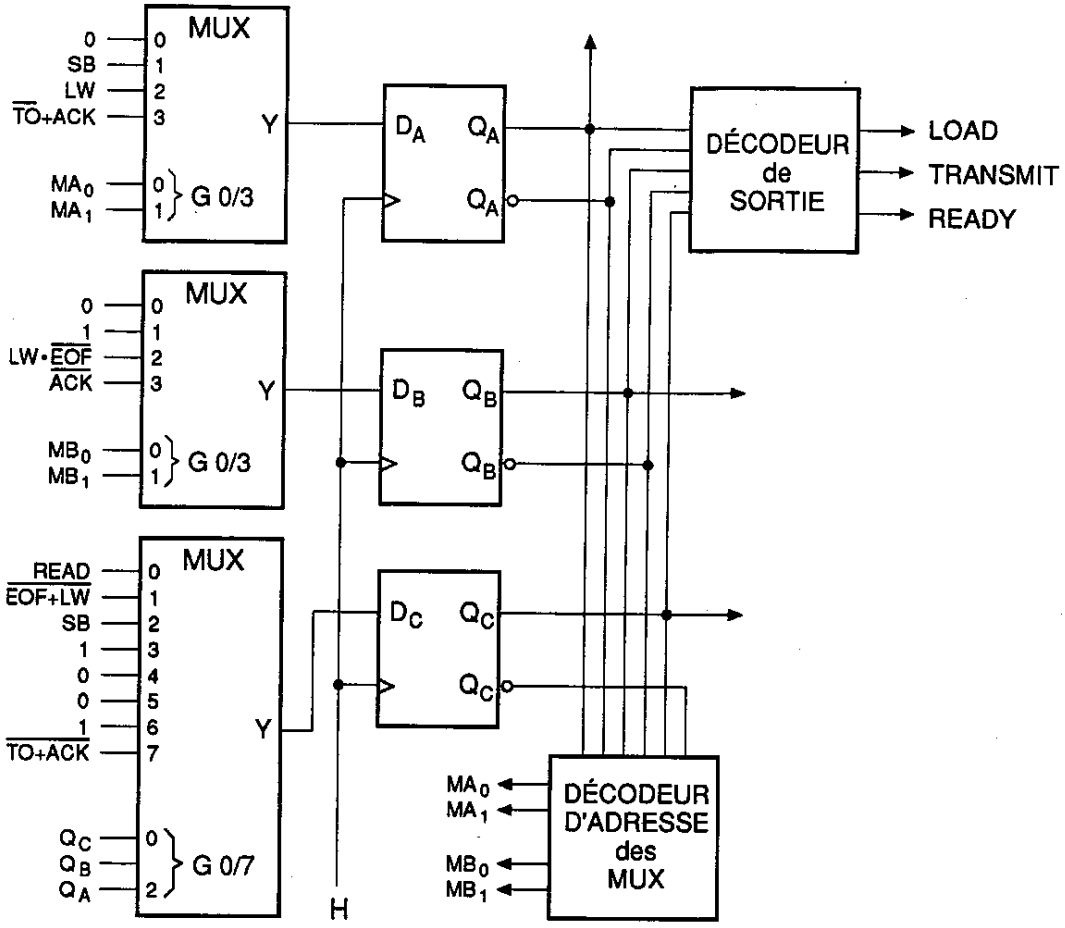
D_C

Q_C \ Q_B	READ	SB	1	0
$\overline{EOF+LW}$	1	$\overline{TO+ACK}$	X	

Table d'adressage des MUX

état	MUX A adresse	MUX A signal	MUX B adresse	MUX B signal	MUX C adresse	MUX C signal
(T_0) 0	0	0	0	$\overline{0}$	2	$\overline{READ}$
(T_1) 1	0	0	2	LW•$\overline{EOF}$	3	$\overline{EOF+LW}$
(T_2) 2	0	0	1	1	4	SB
(T_3) 3	1	SB	1	1	1	1
(T_5) 4	2	LW	1	1	0	0
5	X	X	X	X	X	X
(T_6) 6	0	$\overline{0}$	0	$\overline{0}$	1	$\overline{1}$
(T_4) 7	3	$\overline{TO+ACK}$	3	$\overline{ACK}$	5	$\overline{TO+ACK}$
	↓		↓		↓	
	MUX 4 à 1		MUX 4 à 1		MUX 8 à 1	

donc, dans ce cas, il est préférable de faire un adressage direct car aucune réduction avec adressage indirect



synthèse du décodeur d'adresse des MUX

MA_0

Q_C \ $Q_A Q_B$	00	01	11	10
0	0	0	0	0
1	0	1	1	X

$MA_0 = Q_B Q_C$

MA_1

Q_C \ $Q_A Q_B$	00	01	11	10
0	0	0	0	1
1	0	0	1	X

$MA_1 = Q_A \cdot (\overline{Q}_B \oplus Q_C)$ ou $Q_A\overline{Q}_B + Q_AQ_C$

MB_0

Q_C \ $Q_A Q_B$	00	01	11	10
0	0	1	0	1
1	0	1	1	X

$MB_0 = MA_1 + \overline{Q}_A Q_B$

MB_1

Q_C \ $Q_A Q_B$	00	01	11	10
0	0	0	0	0
1	1	0	1	X

$MB_1 = Q_C \cdot (Q_A \oplus \overline{Q}_B)$ ou $\overline{Q}_BQ_C + Q_AQ_C$

Note : l'adresse du MUX A = $MA_1\ MA_0$
l'adresse du MUX B = $MB_1\ MB_0$

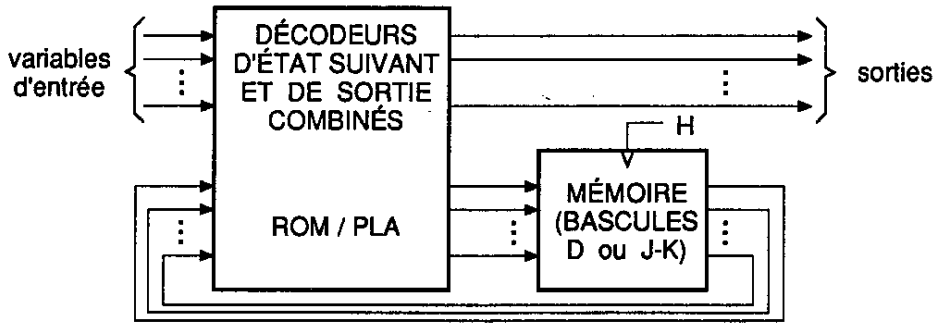
Evidemment, l'avantage de la configuration avec multiplexeurs indirectement adressés n'apparait pas dans cette exemple, mais la réalisation d'un circuit séquentiel comportant 32 états, une dizaine de commandes externes montre que cette configuration simplifie le circuit. (5 MUX 32 à 1 nécessaires lors de la synthèse avec MUX directement adressés; tables de Karnaugh à 5 variables et une dizaine de variables conditionnelles lors de la synthèse classique).

7.6.3 - ROM ou PLA central.

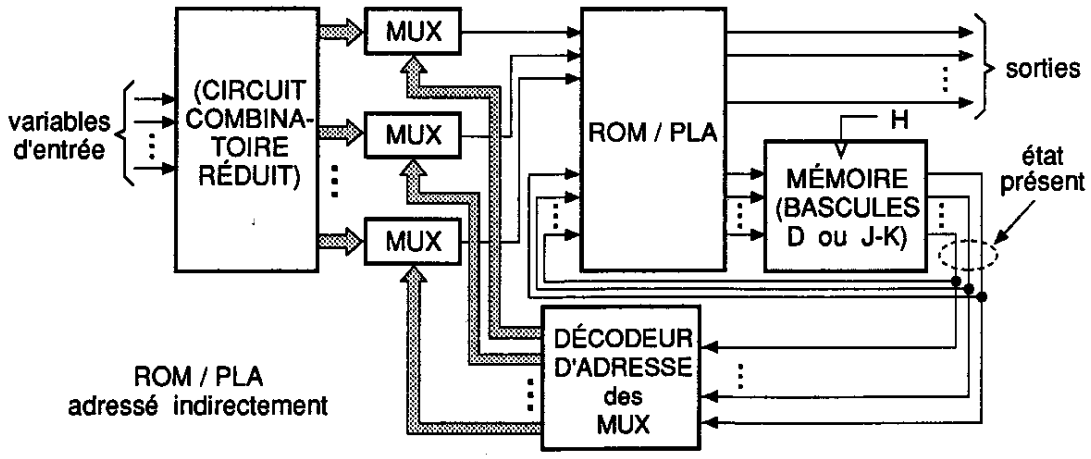
L'avènement des circuits LSI tels les ROM (PROM, EPROM...) et les PLA a modifié l'orientation des design des circuits logiques dont, entre autre, ceux séquentiels synchrones. On peut remplacer aisément les décodeurs d'état suivant et de sortie simultanément avec un ou des ROM / PLA. Les avantages sont immédiats :

- **- simplicité du circuit final**
- **- possibilité de modification du graphe sans toucher au câblage**
- **- synthèse simple**

Tout comme les configurations avec multiplexeurs, celles centrées autour des ROM ou PLA se font avec adressage direct ou indirect. Les figures ci-dessous illustrent les schémas bloc des circuits séquentiels synchrones avec ROM / PLA central. Ces configurations prennent de plus en plus de popularité surtout lors de la synthèse de circuits complexes.



ROM / PLA adressé directement



ROM / PLA
adressé indirectement

Le montage avec ROM adressé directement utilise au maximum la flexibilité des ROM et requiert un minimum de câblage. Cependant, cette flexibilité se paie par la dimension du ROM. En effet, la réalisation d'un graphe de 16 états avec 6 commandes externes exige un ROM ayant une dimension minimum de 2^{10} x n ($10 = 4 + 6$, $2^4 = 16$) ou n est le nombre de sorties + 4. Le prix est moins exigeant dans le cas d'un PLA. Car on fait des réductions à partir des minterms.

Le montage avec ROM indirectement adressé tente d'optimiser l'efficacité du ROM en réduisant les possibilités de branchement à chaque état. Ces possibilités dépendent des combinaisons possibles des variables d'entrée. Or le nombre de branchement à chaque état est souvent beaucoup plus faible que le nombre total de combinaisons disponibles. Pour un état en particulier, les multiplexeurs sélectionnent les conditions de branchement utiles. Les sorties de ces multiplexeurs se réfèrent à des drapeaux et leur nombre est déterminé par le nombre maximum de branchements à partir d'un seul état. Par exemple, la réalisation d'un graphe possédant, dans le pire cas, un état d'où découle 4 transitions possibles requiert 2 drapeaux = 2 sorties de MUX = 2 MUX.

La synthèse d'un circuit séquentiel synchrone suivant ces 2 configurations débute en dressant la table du programme du ROM. Cette table indique pour chacune des adresses possibles, les spécifications des sorties. L'implantation directe de la table suffit puisque le ROM est directement adressé tandis qu'une étape de sélection des conditions de branchement et une synthèse du décodeur d'adresse des MUX sont à prévoir dans l'autre configuration. Si on emploie des PLA, il faut, au préalable, simplifier les expressions des sorties en fonction des commandes externes ou des drapeaux pour chacun des états.

Exemple : système de transmission série. (voir le graphe au paragraphe 7.5.6)

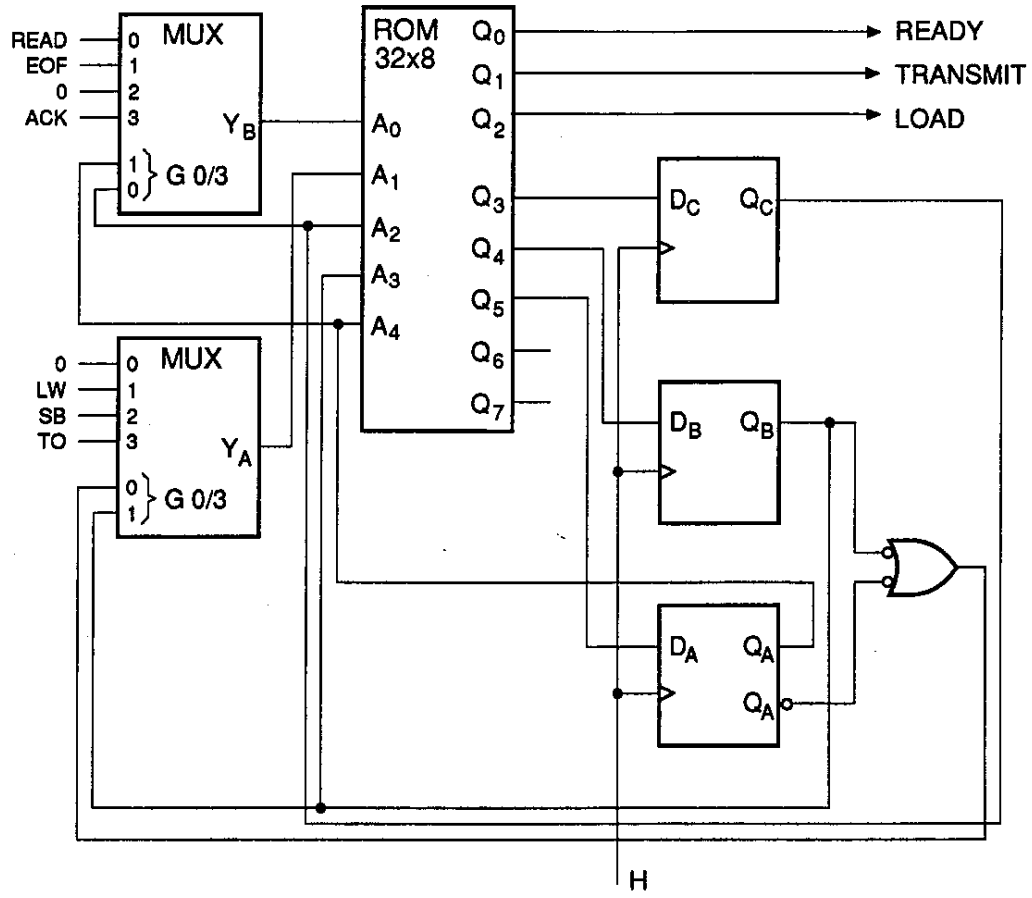
Table de programmation du ROM

état présent			commandes						état suivant			sorties		
Q_A	Q_B	Q_C	READ	EOF	LW	SB	TO	ACK	Q_A	Q_B	Q_C	LOAD	TRANSIT	READY
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			1	0	0	0	0	0	0	0	1	0	0	0
			le reste						X	X	X	X	X	X
0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
			0	0	1	0	0	0	0	1	0	1	0	0
			0	1	X	0	0	0	0	0	0	1	0	0
			le reste						X	X	X	X	X	X
0	1	0	0	0	X	0	0	0	0	1	0	0	1	0
			0	0	X	1	0	0	0	1	1	0	1	0
			le reste						X	X	X	X	X	X
0	1	1	0	0	X	0	0	0	0	1	1	0	1	0
			0	0	X	1	0	0	1	1	1	0	1	0
			le reste						X	X	X	X	X	X
1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
			0	0	1	0	0	0	1	1	0	0	0	1
			le reste						X	X	X	X	X	X
1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	0	0	0	0	0	0	0	1	0	0	1
			le reste						X	X	X	X	X	X
1	1	1	0	0	X	0	0	0	1	1	1	0	0	0
			0	0	X	0	X	1	1	0	0	0	0	0
			0	0	X	0	1	0	0	1	0	0	0	0
			le reste						X	X	X	X	X	X

- Dimension du ROM : 2^9 x 6 = 512 x 6 bits (minimum)
- Avec adressage indirect, on aurait besoin de 2 MUX au maximum, car il n'y a pas plus de 3 possibilités de branchement pour n'importe quel état. Donc la dimension dans ce cas est $2^3 \times 2^2 \times 6 = 32 \times 6$.

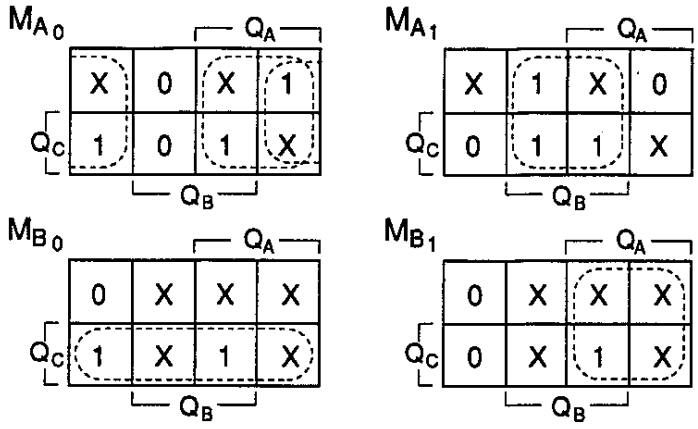
ROM / PLA indirectement adressé.

Dans l'exemple de la transmission série, le maximum de branchements à partir d'un état est 3. On a donc besoin de 2 MUX dont les sorties seront les "drapeaux". Il existe 6 signaux de commande. On peut donc réaliser le système avec 2 MUX 4 → 1 pour sélectionner les conditions de branchement (on décide la taille des MUX après étude du nombre des combinaisons nécessaire des signaux de commande). Le diagramme ci-dessous est un exemple de réalisation.



Calcul du décodeur d'adresses des MUX

état présent $Q_A Q_B Q_C$	signaux qui déterminent l'état suivant	MUX A M_{A_1}	MUX A M_{A_0}	MUX B M_{B_1}	MUX B M_{B_0}
0 0 0	READ	X	X	0	0
0 0 1	LW, EOF	0	1	0	1
0 1 0	SB	1	0	X	X
0 1 1	SB	1	0	X	X
1 0 0	LW	0	1	X	X
1 0 1		X	X	X	X
1 1 0		X	X	X	X
1 1 1	TO, ACK	1	1	1	1



$M_{A_0} = Q_A + \bar{Q}_B$ $\qquad$ $M_{A_1} = Q_B$

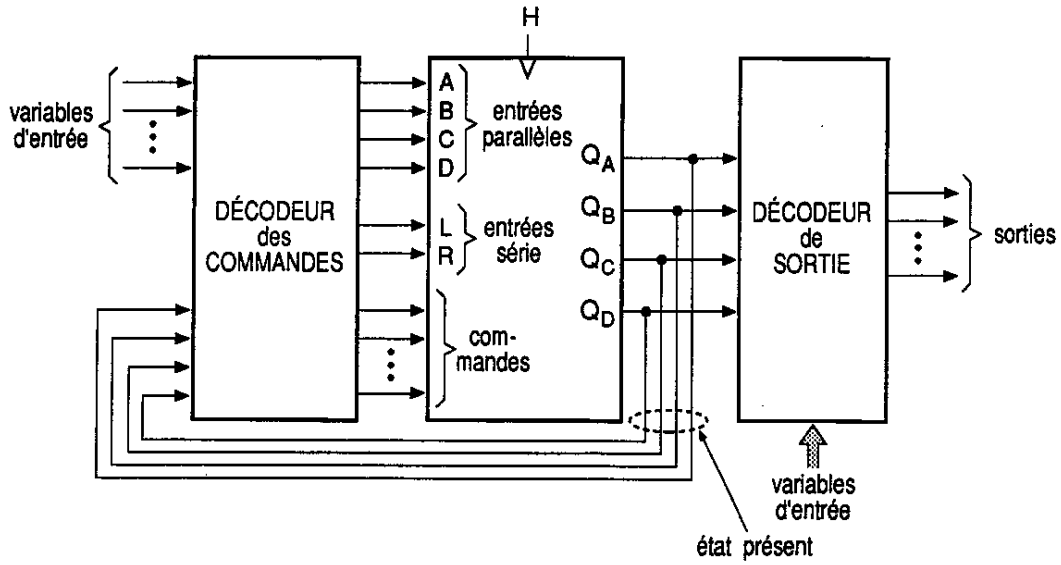
$M_{B_0} = Q_C$ $\qquad$ $M_{B_1} = Q_A$

Table de programmation de la ROM

état présent			drapeaux		état suivant			sorties		
Q_A	Q_B	Q_C	Y_A	Y_B	Q_A	Q_B	Q_C	LOAD	TRANSIT	READY
0	0	0	X	0	0	0	0	0	0	0
			X	1	0	0	1	0	0	0
0	0	1	0	0	0	0	1	1	0	0
			1	0	0	1	0	1	0	0
			X	1	0	0	0	1	0	0
0	1	0	0	X	0	1	0	0	1	0
			1	X	0	1	1	0	1	0
0	1	1	0	X	0	1	1	0	1	0
			1	X	1	1	1	0	1	0
1	0	0	0	X	0	1	0	0	0	1
			1	X	1	1	0	0	0	1
1	0	1	X	X	X	X	X	X	X	X
1	1	0	X	X	0	0	1	0	0	1
1	1	1	0	0	1	1	1	0	0	0
			1	0	0	1	0	0	0	0
			X	1	1	0	0	0	0	0

7.6.4 - Régistre à décalage ou compteur central.

Les techniques démontrées jusqu'à présent utilisaient des éléments "mémoire" du type entrées/sorties parallèles avec bascules D ou JK. Ce paragraphe illustre une approche différente de la synthèse de circuits séquentiels synchrones. Ici, on remplace les éléments "mémoire" par un régistre ou un compteur. Le décodeur d'état suivant devient plutôt un décodeur des commandes logiques du régistre ou du compteur. La figure ci-dessous représente un modèle général avec régistre à décalage ou avec compteur central.



décodeur de commandes : selon l'état présent et les commandes externes, indique au régistre ou au compteur l'action qu'il doit entreprendre, et présente les entrées nécessaires (parallèles ou série).

régistre à décalage au compteur : sert d'éléments "mémoire" pouvant accomplir certaines actions.

décodeur de sortie : identique à celui discuté jusqu'à présent.

a) Régistre à décalage (le 74194).

La construction du décodeur des commandes tient compte évidemment, du type de régistre à décalage utilisé. Le prérequis à la construction consiste à répondre à la question "quelle action doit être entreprise?" Pour le 74194, les actions sont :

mnémonique	action possible
DG1C	*décalage à gauche d'un "1" conditionnel*
DG0C	*" d'un "0" "*
DG1I	*" d'un "1" inconditionnel*
DG0I	*" d'un "0" "*
DD1C	*décalage à droite d'un "1" conditionnel*
DD0C	*" d'un "0" "*
DD1I	*" d'un "1" inconditionnel*
DD0I	*" d'un "0" "*
BC	*branchement conditionnel*
BI	*" inconditionnel*
AC	*aucune action (forcément conditionnel)*

Le mieux à faire est de reprendre le graphe et de faire correspondre un mnémonique à chaque transition.

Étapes :

(1) Remplir une table d'action avec les mnémoniques. La table d'action fournit un outil visuel des actions possibles à effectuer à partir de chaque état.

Étapes suite...

(2) Connaissant les modes d'opération du régistre à décalage, tracer la table de chaque commande du mode d'opération en se basant sur la table d'action.

74194

S_1	S_0	action
0	0	*aucune*
0	1	*décalage à droite*
1	0	*décalage à gauche*
1	1	*chargement*

(3) Remplir une table de donnée pour chacune des entrées série et parallèles du régistre à décalage. Cette étape se base sur la table d'action élaborée précédemment.

Bien entendu, le choix du codage est important et il doit prendre en considération les actions à effectuer. De plus, il ne faut pas perdre de vue que les registres ont une certaine longueur (4 bits pour le 74194). Le nombre de bits de codage est donc égal à la longueur ou à un multiple de la longueur.

La synthèse des expressions obtenues peut se réaliser avec un assemblage de portes ou avec des multiplexeurs. Le choix repose sur la complexité de l'expression.

Exemple : Système de transmission série

Le codage sur 4 bits Q_A = bit à poids fort on peut dresser les tables en prenant seulement les 3 bits de poids faible puisque $Q_A = 0$ dans tous les états

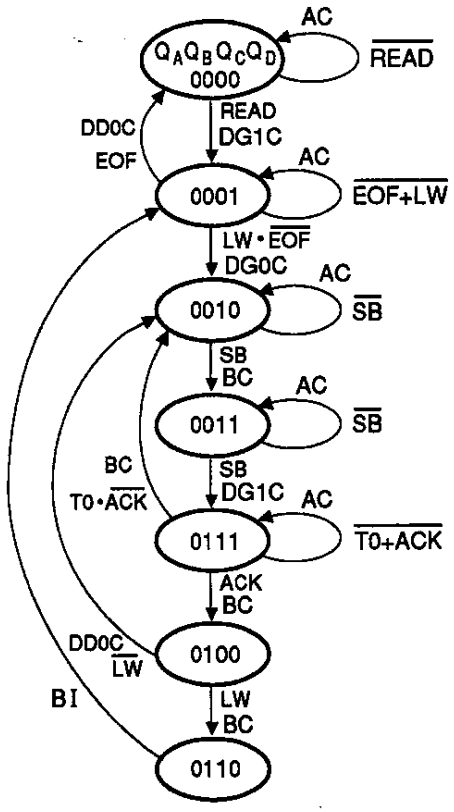


table d'action

Q_D \ Q_B				
	AC + DGIC	AC + BC	BI	BC + DD0C
	AC + DG0C+ DD0C	AC + DGIC	AC + BC	

tables des commandes du mode d'opération

S_1

Q_D \ Q_B				
	READ	SB	1	LW
	$\overline{EOF}$•LW	SB	TO+ACK	X

S_0

Q_D \ Q_B				
	0	SB	1	1
	EOF	0	TO+ACK	X

lorsque : LW = EOF = 0 alors $S_1 = S_0 = 0$ → *aucune action*
LW = 0, EOF = 1 } alors $S_1 = 0, S_0 = 1$ → *décalage à droite*
LW = 1, EOF = 1 }
LW = 1, EOF = 0 alors $S_1 = 1, S_0 = 0$ → *décalage à gauche*

$$S_1 = READ + LW \cdot \overline{EOF} \cdot \overline{Q}_C + TO{+}ACK + Q_B \cdot Q_C \cdot \overline{Q}_D + SB + LW \cdot Q_B \overline{Q}_D$$

$$S_0 = EOF + SB \cdot \overline{Q}_D + TO + ACK + Q_B \overline{Q}_D$$

↳ synthèse meilleure avec MUX

tables des données

- entrées série : lorsqu'on demande un décalage

L

Q_D \ Q_B				
	1	X	X	X
	0	1	X	X

R

Q_D \ Q_B				
	X	X	X	0
	0	X	X	X

décalage à gauche d'un "1"

décalage à gauche d'un "0"

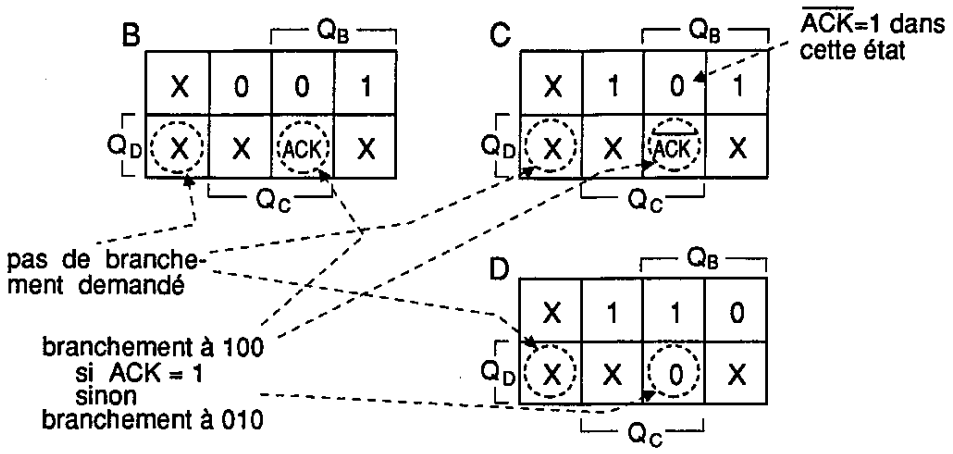
pas de décalage à droite demandé

décalage à droite d'un "0"

$$L = \overline{Q}_D + Q_C$$

$$R = 0$$

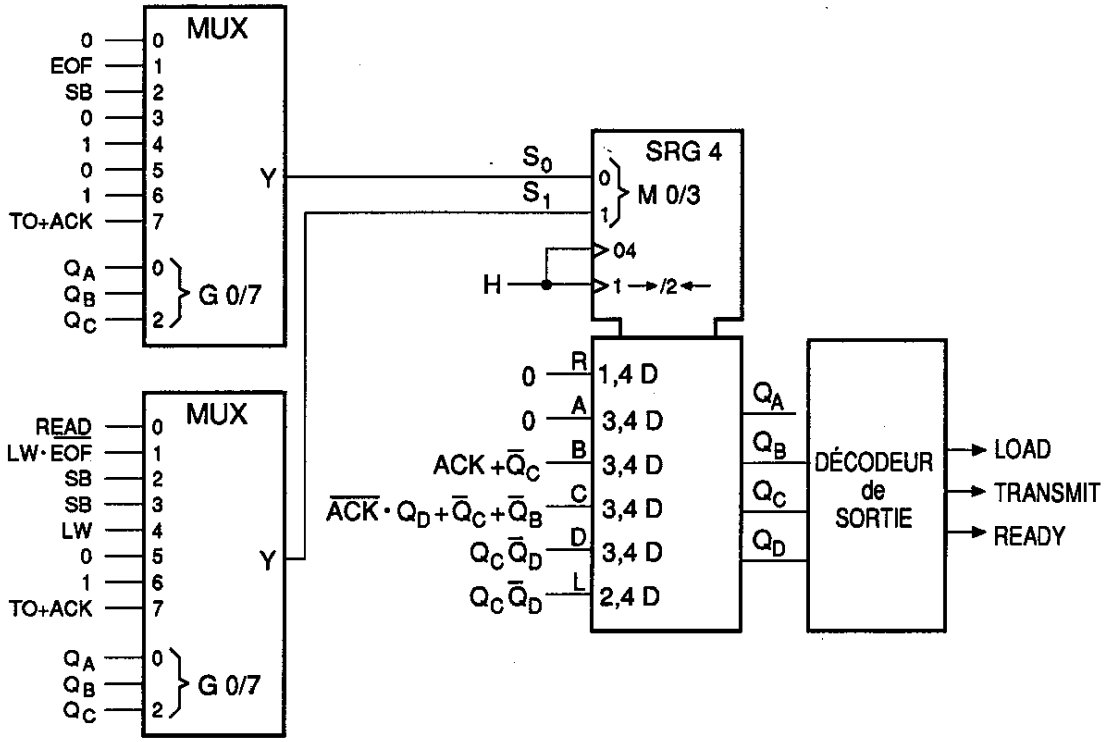
- entrées parallèle : lorsqu'on demande un branchement



$$B = ACK + \overline{Q}_C$$

$$C = \overline{ACK} \cdot Q_D + \overline{Q}_C + \overline{Q}_B$$

$$D = Q_C \cdot \overline{Q}_D$$



b) Compteur (le 74163).

Similairement au montage avec registre à décalage central, on doit ajouter une information complémentaire au graphe qui indique l'action entreprise pour cette transition par le biais d'un mnémonique. L'action

est ici plus limitée :

- ***continuité de la séquence du compte***
- ***branchement par chargement, bris de la séquence***

mnémonique	action possible
CC	*Comptage conditionnel*
CI	*" inconditionnel*
BC	*branchement conditionnel*
BI	*" inconditionnel*
AC	*aucune action*

Les étapes de synthèse avec compteur central ressenmblent à celles décrites avec régistre à décalage.

Étapes :

(1) Remplir la table d'action avec les 4 mnémoniques.

(2) Connaissant les modes d'opération du compteur, tracer la table de chaque commande du mode d'opération.

74163 "LOAD" → pour le branchement
"ENP" → pour le comptage
"ENT" pour le comptage
attention: "LOAD" prioritaire sur "ENP", "ENT"

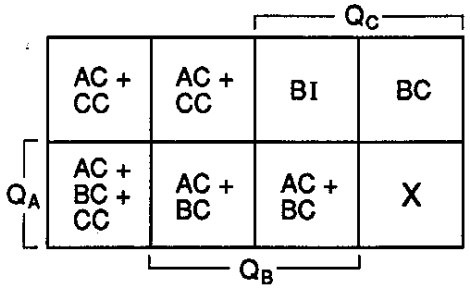
(3) Remplir les tables de données des entrées parallèles.

Le codage le plus efficace est celui où la séquence principale des états du graphe suit la progression binaire pour permettre au compteur de compter normalement. Ainsi, chaque transition sautant un un état ou revenant vers un état précédent correspond à un branchement. Encore ici, le nombre de bits de codage doit être égal à un multiple de la longueur du compteur.

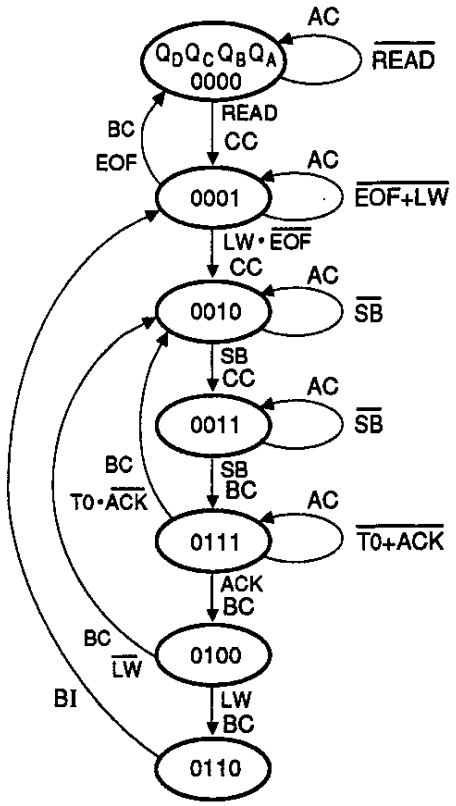
Exemple : Système de transmission série

Avec 74163
codage sur 4 bits
Q_D = bits de poids fort ⟵ attention

table d'action

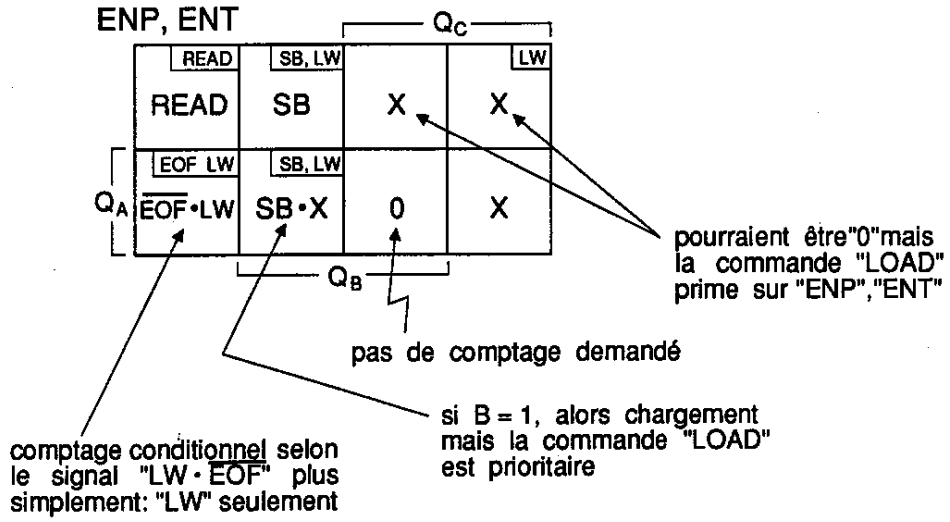


(74163 suite...)



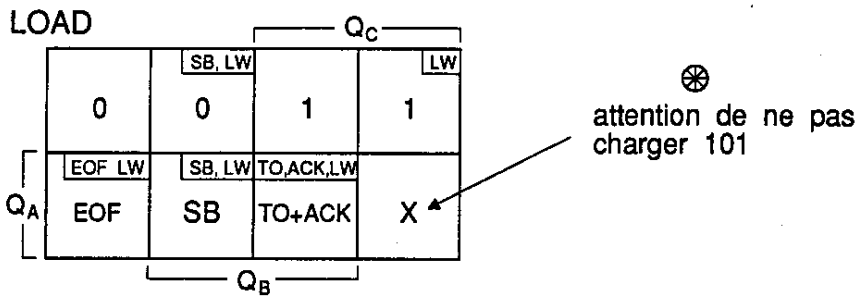
tables des commandes du mode d'opération

- "ENP" et "ENT" : lorsqu'on demande un comptage



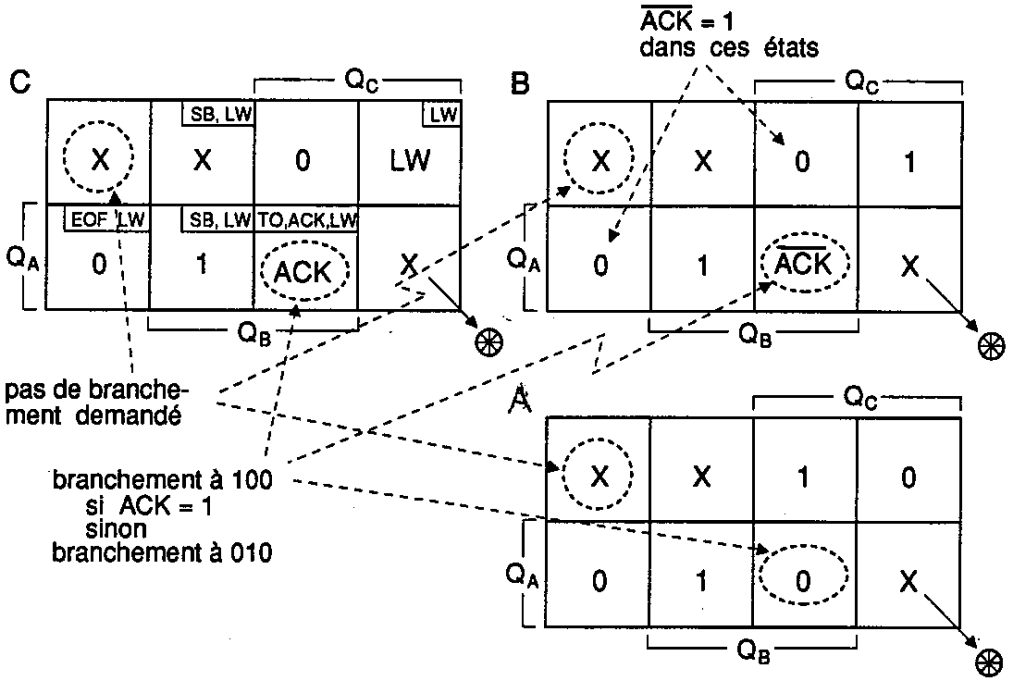
$$ENP = ENT = READ + \underbrace{LW\ \overline{EOF} \cdot \overline{Q}_B}_{\text{ou } LW \cdot \overline{Q}_B} + SB$$

- "LOAD" : lorsqu'on demande un branchement



$$LOAD = EOF + SB \cdot Q_A + TO + ACK + \overline{Q}_A \cdot Q_C$$

tables de données



$$C = ACK + LW \cdot \overline{Q}_A + \overline{Q}_C \cdot Q_B$$

$$B = \overline{ACK} \cdot Q_B \cdot Q_A + \overline{Q}_C \cdot Q_B + Q_C \cdot \overline{Q}_B = \overline{ACK} \cdot Q_B \cdot Q_A + Q_C \oplus Q_B$$

$$A = \overline{Q}_C \cdot Q_B + Q_B \cdot \overline{Q}_A$$

Effect du codage

Le codage affecte particulièrement la synthèse avec régistre à décalage au compteur central. Un codage indifférent de la configuration employée risque de compliquer la synthèse et la grosseur du circuit.

Exemple : Système de transmission série avec un autre codage tenant compte de la configuration avec compteur central, qui est un compteur binaire d'où le code binaire utilisé (pour le registre à décalage, un code décalé est préférable).

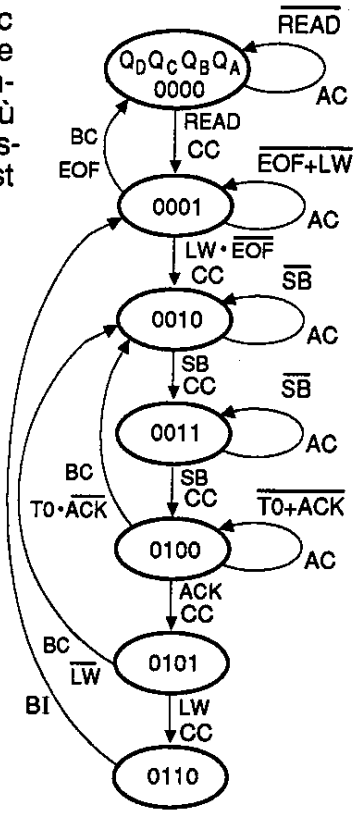


table d'action

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	AC + CC	AC + CC	BI	AC + BC + CC
1	AC + CC + BC	AC + CC	X	BC + CC

tables des commandes du mode d'opération

ENP

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	READ	SB	X	ACK
1	LW • $\overline{EOF}$	SB	X	LW

LOAD

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	0	0	1	TO • $\overline{ACK}$
1	EOF	0	X	$\overline{LW}$

$ENP = READ + LW \cdot \overline{EOF} \cdot \overline{Q}_B Q_A + SB + ACK$

$LOAD = EOF + TO \cdot \overline{ACK} + \overline{LW} \cdot Q_C Q_A + Q_B \cdot Q_C$

tables de donnée

C

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	X	X	0	0
1	0	X	X	0

B

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	X	X	0	1
1	0	X	X	1

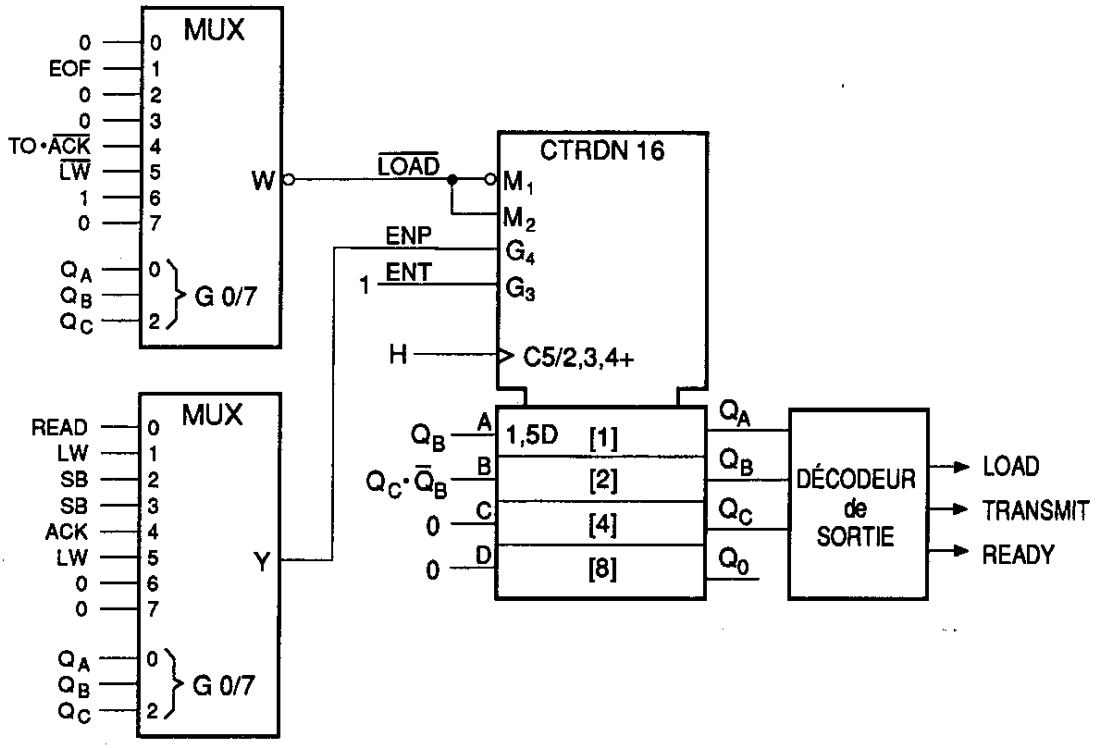
A

Q_A \ Q_B	Q_C = 0	Q_C = 0	Q_C = 1	Q_C = 1
0	X	X	1	0
1	0	X	X	0

$C = 0$

$B = Q_C \overline{Q}_B$

$A = Q_B$

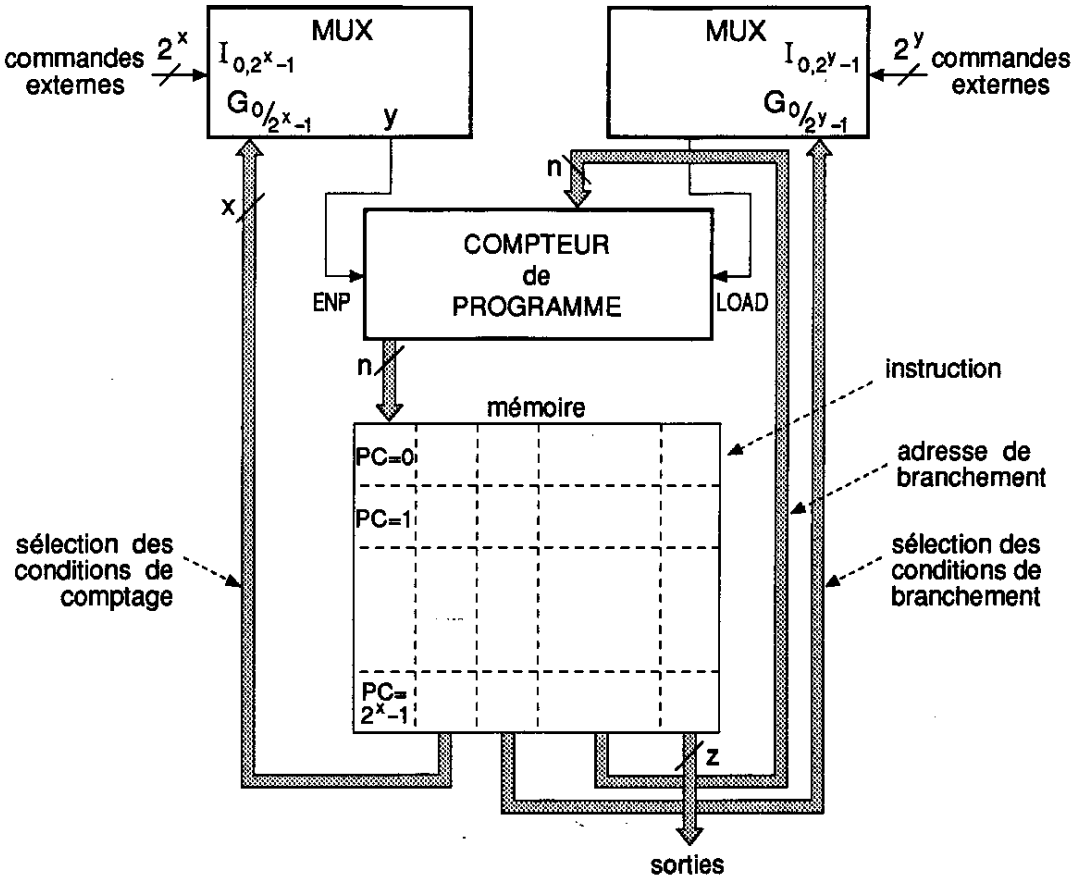


7.7 - Systèmes à micro-instructions (microcontrôleurs).

7.7.1 - Architecture de départ.

Les circuits séquentiels synchrones avec micro-instructions procurent l'avantage d'une souplesse accrue : on peut remodeler complètement le graphe en inscrivant tout simplement un nouveau contenu dans une mémoire (ROM, PLA ou RAM). Chaque mot enregistré dans la mémoire est une instruction machine qui contient un code d'opération (OPCODE) et les paramètres utiles à l'opération tels la sélection des conditions de comptage ou de branchements, l'adresse de branchement, etc... Le tout forme un programme, semblable à celui rencontré dans un ordinateur par exemple. Chaque instruction est exécutée à tour de rôle suivant le plan du graphe.

Pour ce faire, le module programmable (la mémoire) est arrangé en matrice où chaque ligne forme une instruction. La position de la ligne dans la matrice est est l'adresse de l'instruction à être exécutée dans le programme. On l'appelle, à juste titre, le "program counter" (PC).



Plus spécifiquement, un système à micro-instructions doit permettre les opérations de base suivantes :

* *incrément du compteur de programme (PC←PC + 1) pour pointer l'instruction suivante dans la mémoire indépendamment des commandes externes.*

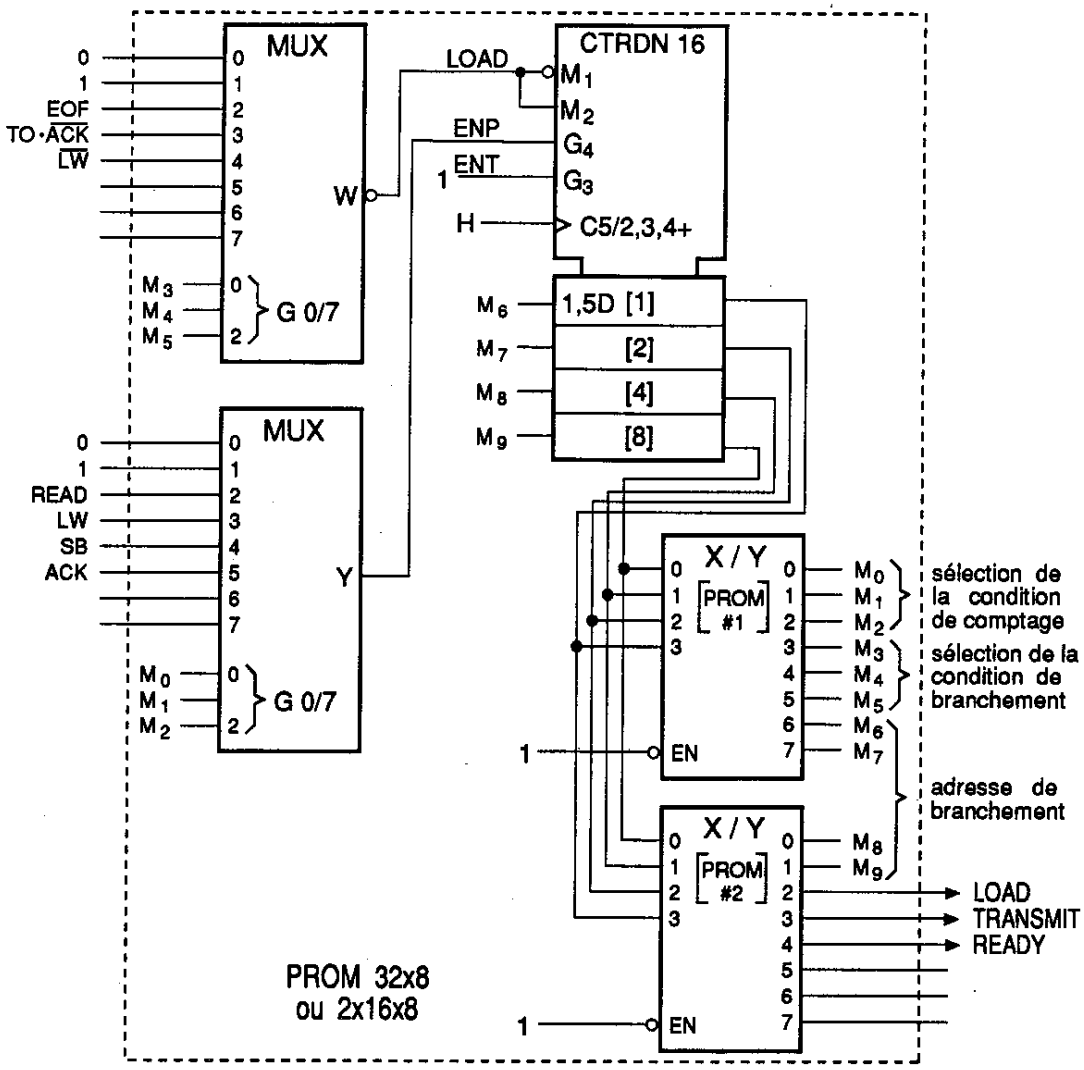
* *incrément du compteur de programme conditionnellement aux commandes externes*

* *chargement du compteur de programme (PC ← adresse de branchement) pour une rupture dans la séquence normale indépendamment des commandes externes*

* *chargement du compteur de programme conditionnellement aux commandes externes.*

* *incrément ou chargement du compteur de programme dépendamment des commandes externes.*

Exemple : Système de transmission série
(voir le graphe de l'exemple précédent avec codage optimal)



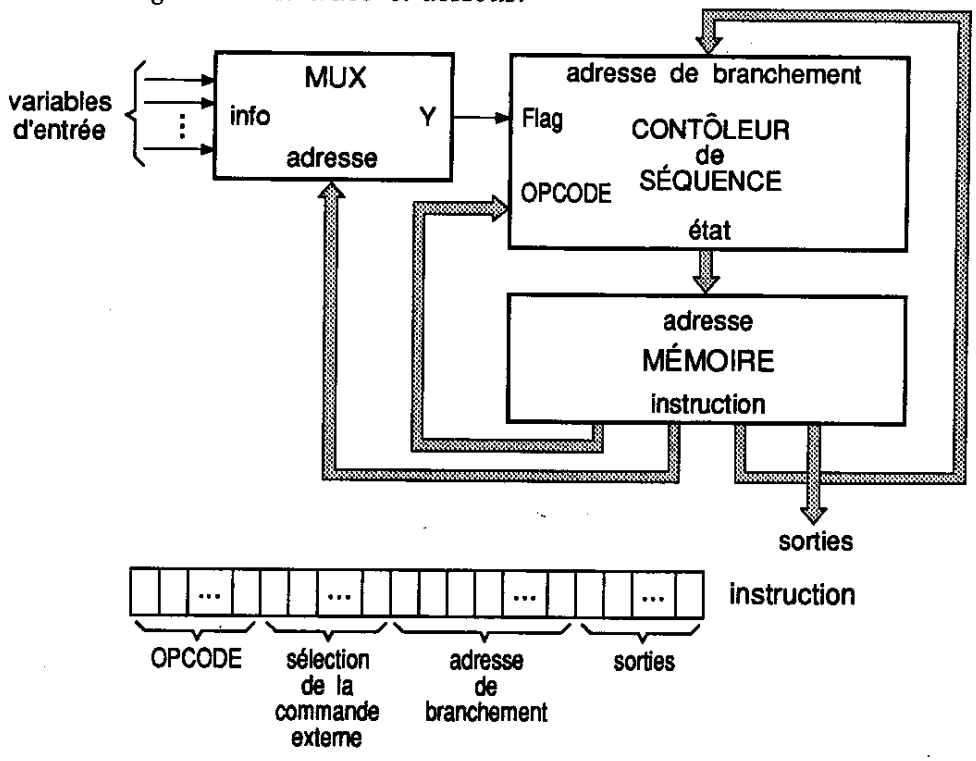
Programme

ADRESSE						SORTIES			ADRESSES branchement				CONDITIONS branchement			comptage		
Q_C	Q_B	Q_A	M_{15}	M_{14}	M_{13}	M_{12}	M_{11}	M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
0	0	0	X	X	X	0	0	0	X	X	X	X	0	0	0	0	1	0
0	0	1	X	X	X	0	0	1	0	0	0	0	0	1	0	0	1	1
0	1	0	X	X	X	0	1	0	X	X	X	X	0	0	0	1	0	0
0	1	1	X	X	X	0	1	0	X	X	X	X	0	0	0	1	0	0
1	0	0	X	X	X	0	0	0	0	0	1	0	0	1	1	1	0	1
1	0	1	X	X	X	1	0	0	0	0	1	0	1	0	0	0	1	1
1	1	0	X	X	X	1	0	0	0	0	0	1	0	0	1	0	0	0
1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

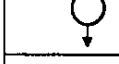
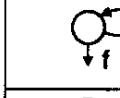
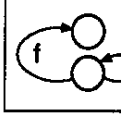
7.7.2 - Ensemble fixe de micro-instructions

L'architecture du système avec micro-instruction peut-être travaillé dans le but d'encapsuler dans un seul boîtier un opérateur séquentiel synchrone, un contrôleur de séquence. La simplicité de programmation étant un critère, la transformation aboutit à un contrôleur de séquence avec un ensemble fixe d'instructions machines. Ceci implique donc que la sélection de l'opération désirée apparaisse dans l'instruction sous la forme de l' "OPCODE".

Le schéma général est tracé ci-dessous.

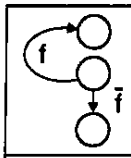
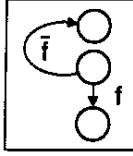


EXEMPLE : réalisation d'un contrôleur de séquence simple avec compteur central.

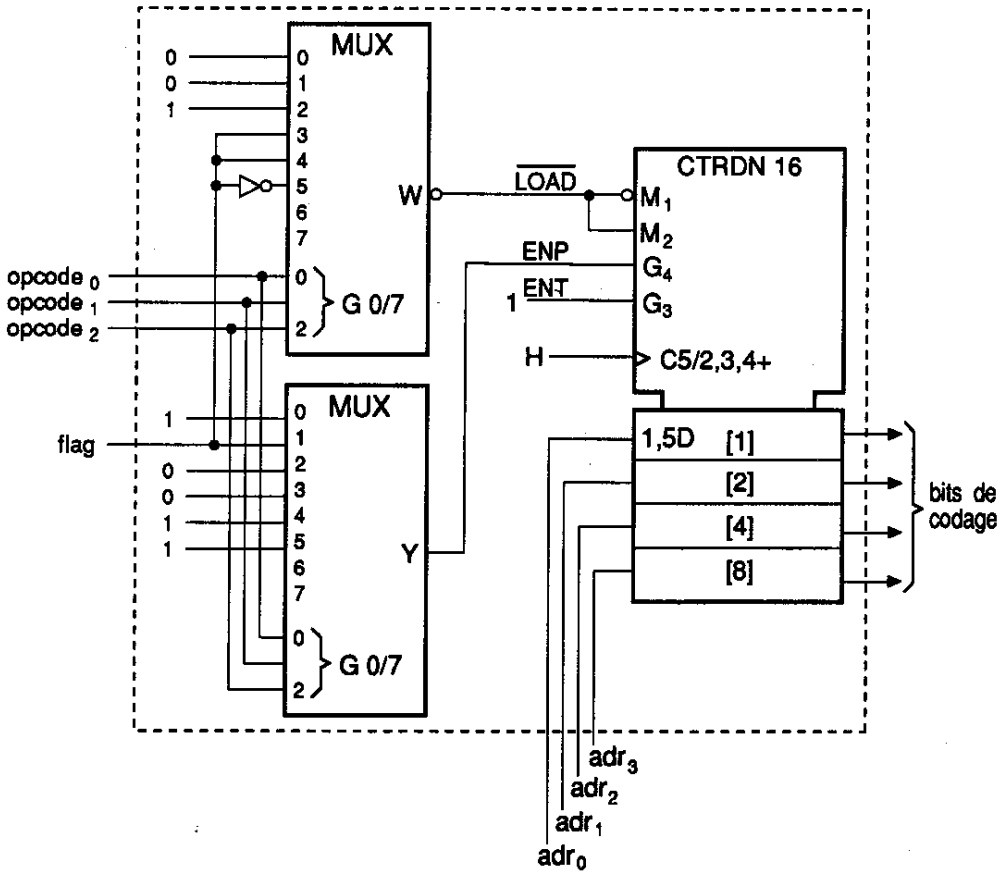
	mnémonique	opcode	instruction
	II	000	incrémente inconditionnellement (PC ← PC+1)
	MIC	001	maintient ou incrémente conditionnellement si (flag=0) alors (PC ← PC) sinon (PC ← PC+1)
	BI	010	branche inconditionnellement (PC ← adr)
	MBC	011	maintient ou branche conditionnellement si (flag=0) alors (PC ← PC) sinon (PC ← adr)

(continue...)

EXEMPLE (suite...) réalisation d'un contrôleur de séquence simple avec compteur central.

	mnémonique	opcode	instruction
	IBC	100	incrémente ou branche conditionnellement si (flag=0) alors (PC ← PC+1) sinon (PC ← adr)
	BIC	101	branche ou incrémente conditionnellement si (flag=0) alors (PC ← adr) sinon (PC ← PC+1)

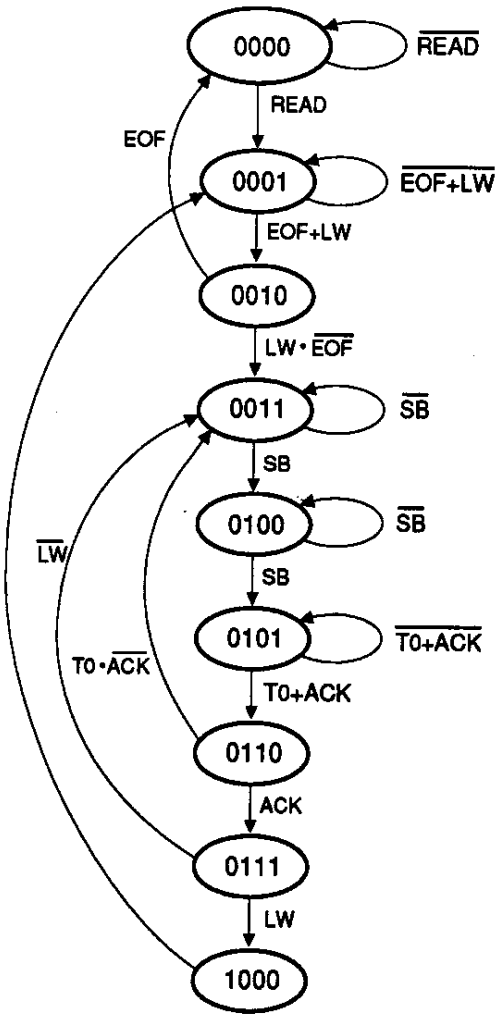
cet ensemble d'instructions est nettement suffisant pour résoudre les graphes dont le codage est optimal pour une synthèse avec la configuration du compteur central.



Cette structure de base se retrouve dans plusieurs contrôleurs de séquences encapsulés. Évidemment, certaines améliorations ont été apportées tels une pile ("stack"), sous-routines, branchements relatifs, etc... Le 74S482 est un exemple de contrôleur de séquence (aussi le 8x02 de "Signetics", le MC2909 de "Advanced micro devices").

Dans la structure à micro-instructions, l'adresse de branchement est continue dans l'instruction. Il faut donc organiser le graphe de telle sorte que, à partir d'un état on ne puisse brancher que vers un et seul état. Dans la structure avec ensemble fixe de micro-instructions, il faudra en plus qu'à chaque état il n'existe que deux possibilités pour l'état suivant (maintient, incrémentation ou branchement); chaque possibilité dépendant de la valeur du flag.

Afin de s'adapter à ces conditions, le graphe du système de transmission série deviendrait le suivant :



CHAPITRE 8

MACHINES SÉQUENTIELLES ALGORITHMIQUES

8.1 - Introduction

8.1.1 - Définition

Il arrive parfois que les spécifications d'un circuit à concevoir soient telles qu'on ne voit pas comment on pourrait esquisser un graphe primitif. C'est que la synthèse à partir d'un diagramme d'état rencontre une limite lors de la conception des circuits de traitements de données. Ces derniers circuits se composent, en fait, d'un nombre élevé de signaux de commande ou d'entrée; une approche de design de circuits séquentiels synchrones aboutit à la résolution d'une table d'état /présent contenant trop de possibilités. La conception d'un système qui multiplierait 2 nombres de 4 bits par additions successives est un exemple (ex: circuit arithmétique "Add & Shift", chapître 6).

La manière d'aborder un problème impliquant un traitement de données consiste à élaborer, dans une première étape, un algorithme dit matériel (par opposition à l'algorithme logiciel rencontré en informatique). L'algorithme définit la séquence des actions à entreprendre pour obtenir la solution exigée par le problème. L'algorithme matériel indique la procédure à suivre permettant l'implantation d'une solution au problème avec certaines pièces d'équipement.

La création de l'algorithme matériel nécessaire comme base de départ de quelques types de circuits est, à l'origine de leur nom : les machines séquentielles algorithmiques (MSA).

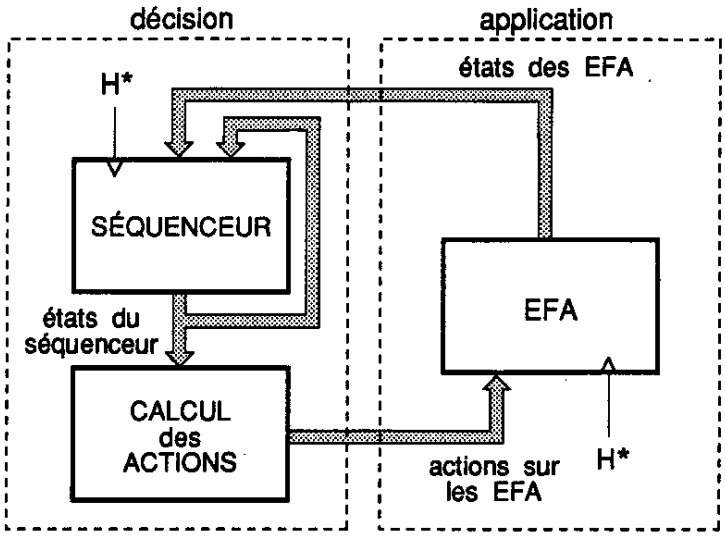
8.1.2 - Principes.

La structure générale des MSA se divise en 2 sous-ensembles distincts :

1 - *les éléments fonctionnels de l'application (EFA) spécifiés par une analyse approfondie du problème. Ce sont les EFA qui agissent et produisent les actions demandées par l'algorithme.*

2 - *le séquenceur qui pilote les EFA. Il génère une horloge multi-phases programmable, laquelle valide l'action entreprise. Le séquenceur dirige donc le système à travers l'algorithme.*

La figure qui suit montre la structure classique des MSA ; remarquez les 2 sous-ensembles et leurs interconnexions.



* l'horloge se rend directement à toutes les entrées de synchronisme des éléments de la MSA.

Suivant la position de la machine dans l'algorithme, le séquenceur détermine la prochaine position et ce, tenant compte :

a) de l'état présent (position immédiate)
b) des états des EFA (les signaux produits par les EFA)

À chaque position possible dans l'algorithme correspond un état du séquenceur et, à chaque état du séquenceur est activée une et une seule phase. L'ensemble des phases générées par le séquenceur et l'ensemble des états des EFA produisent des signaux qui commande les EFA eux-mêmes par le biais des commandes synchrones des éléments des EFA. Les phases ont ici un rôle de validation en ce sens qu'elles permettent l'action produite par la commande lorsque la machine est dans la bonne position.

Il faut donc se demander pour chacune des commandes synchrones des éléments des EFA

- dans quelles positions de l'algorithme doit elle être activée
- sous quelles conditions

MSA → éléments fonctionnels de l'application
= exécutent les actions décrites par l'algorithme

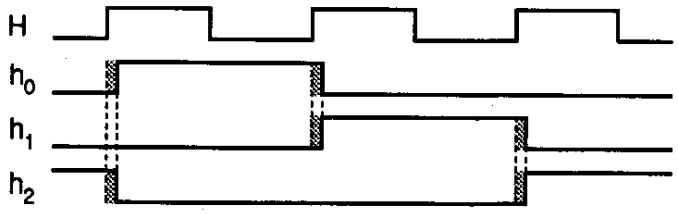
MSA → séquenceur
= établit la marche à suivre à travers l'algorithme

8.2 - Phases jointes ou disjointes.

Le séquenceur d'une MSA est susceptible de produire 2 types de phases : 1) les phases à temps jointifs (phases jointives), et 2) les phases à temps disjoints (phases disjointes). Il importe de connaître comment produire ces phases et sélectionner le bon type.

8.2.1 - Phases jointives.

Le diagramme temporel des phases jointives montre un chevauchement possible d'une phase avec la suivante.



▒ : chevauchements des phases

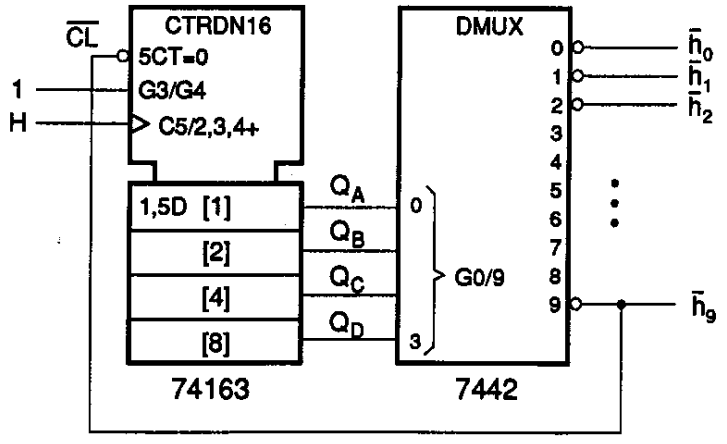
horloge 3 phases non-programmable (la séquence des phases se répète sans interrruption de la séquence)

Les phases jointives sont des "blips" synchrones. Elles sont synchronisées par l'horloge mère et possèdent donc un léger décalage temporel entre le front montant de l'horloge mère et leur propre front montant.

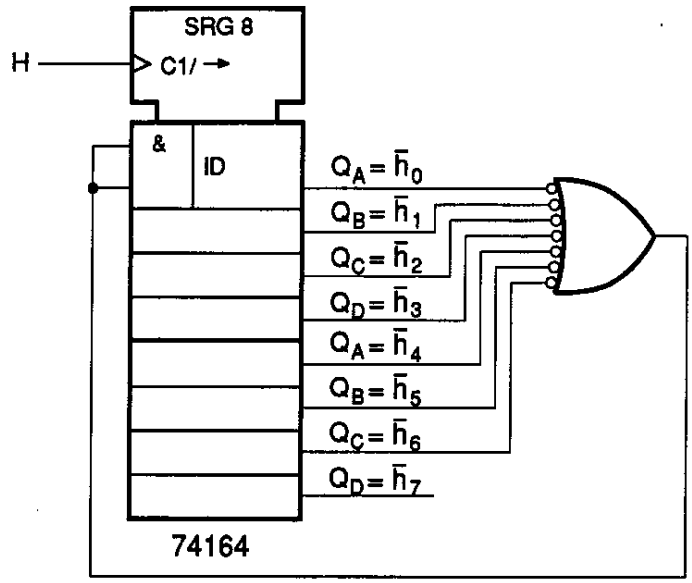
Chaque phase étant un signal synchrone, elle n'est pas exempte des aléas dus aux délais des circuits combinatoires. Elle peut donc être utilisée sur toutes les commandes synchrones mais son emploi devient dangereux sur des entrées asynchrones (qui ne doivent d'ailleurs pas être utilisées dans tout circuit séquentiel synchrone "clean"), comme signal d'horloge H ou sur des commandes d'inhibition des bascules "latch D" (voir paragraphe 4.4.2) et RAM.

phases jointives = *signaux synchrones*
≠ *signal d'horloge*
≠ *commandes d'inhibition*
- bascules "latch D"
- RAM

Exemple #1 : compteur binaire et décodeur



Exemple #2 : avec régistre à décalage circulant un "0"



8.2.2 - Phases disjointes.

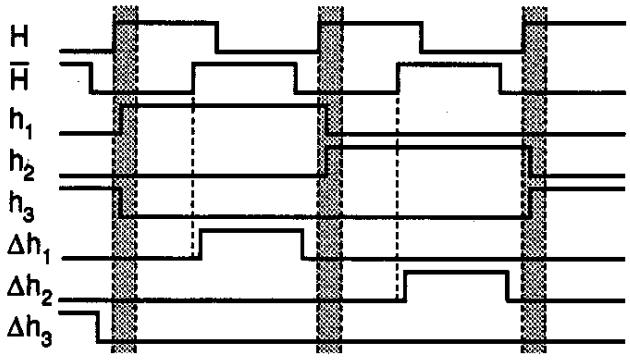
Les phases disjointes ne servent que lorsque les phases jointives ne conviennent pas. On les rencontre donc comme

- commandes de synchronisme ($\neq$ H) sur certains boitiers tels le 74193 (maître-esclave)
- commandes d'inhibition sur "latch D" et RAM seulement.

Les phases disjointes sont des signaux sans aléa mais, en contrepartie, elles sont désynchronisées par rapport à l'horloge mère du système. L'élimination des aléas se réalise simplement en échantillonnant avec $\overline{H}$ en avance sur H (paragraphe 4.11.1). Ainsi, $\overline{H}$ précède la zone perturbée qui suit immédiatement la montée de H indiquant un changement possible des valeurs des variables indépendantes.

La règle pour produire les phases disjointes se résume finalement à échantillonner les phases jointives avec $\overline{H}$ en avance sur H.

Exemple :

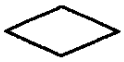


: zones dangereuses contaminées par les aléas

8.3 - Organigramme MSA.

L'organigramme MSA est un organigramme conçu spécialement pour la description d'opérations séquentielles dans un système logique. Il se compose exclusivement de 4 éléments de base : la boite de départ ou d'arrivée, la boite d'état, la boite de décision et la boite conditionnelle.

organigramme MSA →
- → boite de départ ou d'arrivée 

- → boite d'état 
- → boite de décision 
- → boite conditionnelle 

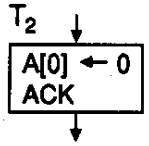
1) boite d'état

La boite d'état, de forme carrée, contient une série d'opérations sur les régistres (une bascule est un régistre de longueur unité) et sur l'activation de quelques sorties. Elle porte le nom "d'état" car elle est associée avec un des états du séquenceur, lequel est identifié à l'extérieur, en haut et à gauche de la boite.

symbolique des opérations	description
A[x+b-1,x] ← B[y+b-1;y]	transfert de b bits du régistre B vers A
A[x+b-1;x] ← 0	mise à 0 de b bits du régistre A
A[ ] ← A[ ]+1	incrémentation du contenu du régiatre A
A[ ] ← A[ ]+B[ ]	addition du contenu du régistre B à A
A[n-1;1] ← A[n-2;0] A[0] ← 0	décalage à gauche du régistre A (A=n bits) avec un "0"
A[n-2;0] ← A[n-1;1] A[n-1] ← 1	décalage à droite du régistre A (A=n bits) avec un "1"
A[ ] | B[ ]	concaténation des régistres A et B

note : A[x-1;0] = les x bits les moins significatives du régistre A
A[0] = la LSB de A
A[ ] = le contenu du régistre A
pour une bascule, ne pas mettre de crochets après la variable d'identification.

Exemple :

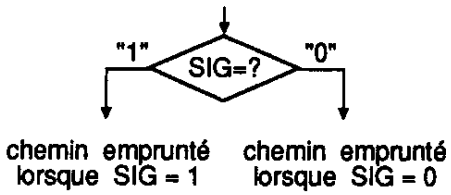


Lorsque le système entre dans l'état "T_2", il met à "0" le régistre A et il active le signal de sortie "ACK".

2) boite de décision

La boite de décision indique deux ou plusieurs cheminements possibles. Le chemin emprunté dépend de la valeur du signal inscrit à l'intérieur de la boite.

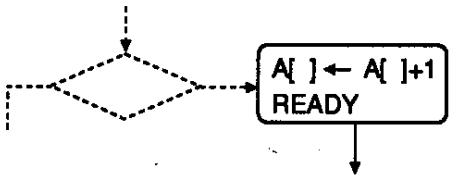
Exemple :



3) boite conditionnelle

La boite d'état et la boite de décision sont familières à ceux dessinant des organigrammes en informatique. Par ailleurs, la boite conditionnelle est unique à l'organigramme MSA. Elle se différencie de la boite d'état par ses coins arrondis. La voie d'entrée provient directement d'une des sorties d'une boite de décision. On indique, à l'intérieur de la boite, l'opération à effectuer sur les régistres et les signaux de sortie à activer.

Exemple :

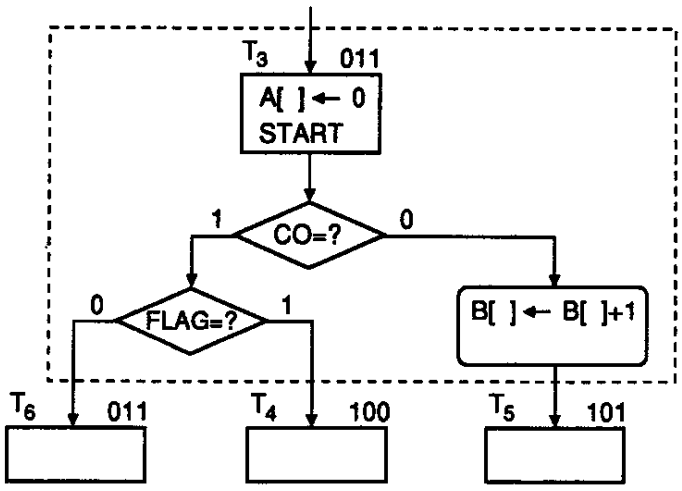


Définition de bloc :

Un bloc est la structure délimitant un état du séquenceur. Il consiste en une boite d'état à laquelle se greffent toutes boites de décision et conditionnelles connectées à sa sortie. Ainsi un bloc n'a qu'une seule voie d'entrée mais peut avoir plusieurs chemins de sortie.

L'organigramme MSA consiste donc en plusieurs blocs inter-reliés.

Exemple :



la ligne pointillée encercle un bloc

Attention : Un même régistre ne doit pas apparaître 2 fois, d'un côté comme de l'autre d'une assignation, dans un bloc. Les variables servant de test de décision ne doivent pas être influencées par les opérations apportées plus tôt dans le même bloc. Advenant ces cas, il faut scinder le bloc en autant de parties que nécessaire pour respecter ces règles.

Le problème se limite à bien définir le type de boîte notamment dans le cas: boite d'état vs boite conditionnelle.

N'oubliez pas qu'un bloc indique l'ensemble des actions à suivre dans un état du séquenceur.

8.4 - Synthèse.

On pourrait définir brièvement la démarche à respecter lors de la synthèse d'une machine séquentielle algorithmique. Mais le gros du travail se résume à une bonne analyse du problème mettant en évidence les éléments nécessaires au bon fonctionnement du système.

Les étapes sont :

(a) *Établir l'algorithme de la machine.*

(b) *Traduire l'algorithme en organigramme MSA.*

Cette étape impose réflexion :

l'organigramme MSA fait appel à des actions limitées sur régistres. On doit donc obtenir l'organigramme et définir les EFA en même temps. Les EFA découlent de l'organigramme; l'organigramme sélectionne les EFA.

L'expérience réduit la difficulté de cette étape. C'est pourquoi plusieurs exemples de MSA composent ce chapître.

(c) *Définir tous les éléments fonctionnels de l'application essentiels. Faire les interconnexions entre chacun.*

(d) *Traduire l'organigramme en graphe.*

L'organigramme MSA séparé en blocs est similaire à un graphe. Chaque bloc = un état.

(e) *Faire la synthèse du séquenceur suivant le graphe.*

(f) *Définir la condition d'activation de chacune des commandes externes des EFA.*
Faire la synthèse des expressions obtenues.

Les conditions d'activation sont fonction de l'état dans lequel ils doivent être actifs. Le séquenceur produit les signaux nécessaires à cette fin (les phases).

Il faut se rappeler que dans certaines circonstances, les phases disjointes sont obligatoires.

8.5 - Exemples.

8.5.1 - Multiplication par additions successives.

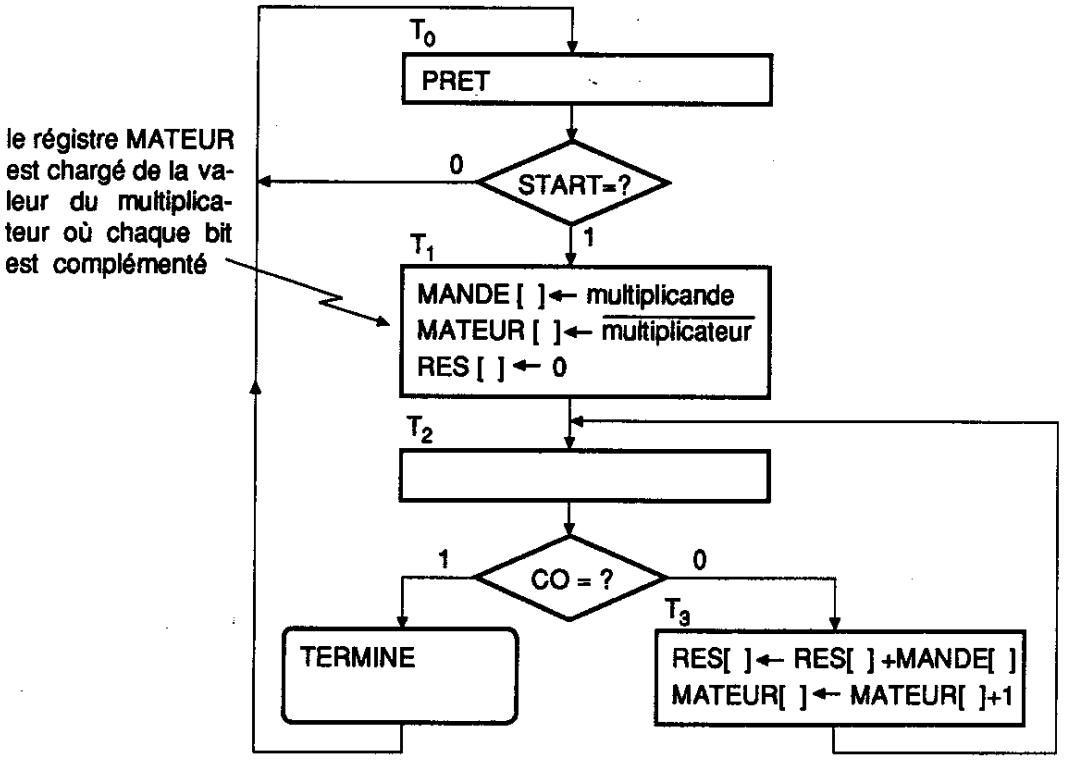
C'est la multiplication suivant le principe le plus simple :

additionner le multiplicande autant de fois que la valeur du multiplicateur l'exige. On suppose un signal de départ "START" ne pouvant arriver que dans l'état de repos initial.

Algorithme :

```
program muladds
    begin
    res: = 0 ;
    obtient (mande, mateur) ;
(*mande = multiplicande, mateur = multiplicateur*)
    while mateur > 0 do
        begin
        res: = res + mande ;
        mateur : = mateur - 1
        end ;
    affiche (res)
    end.
```

Organigramme :

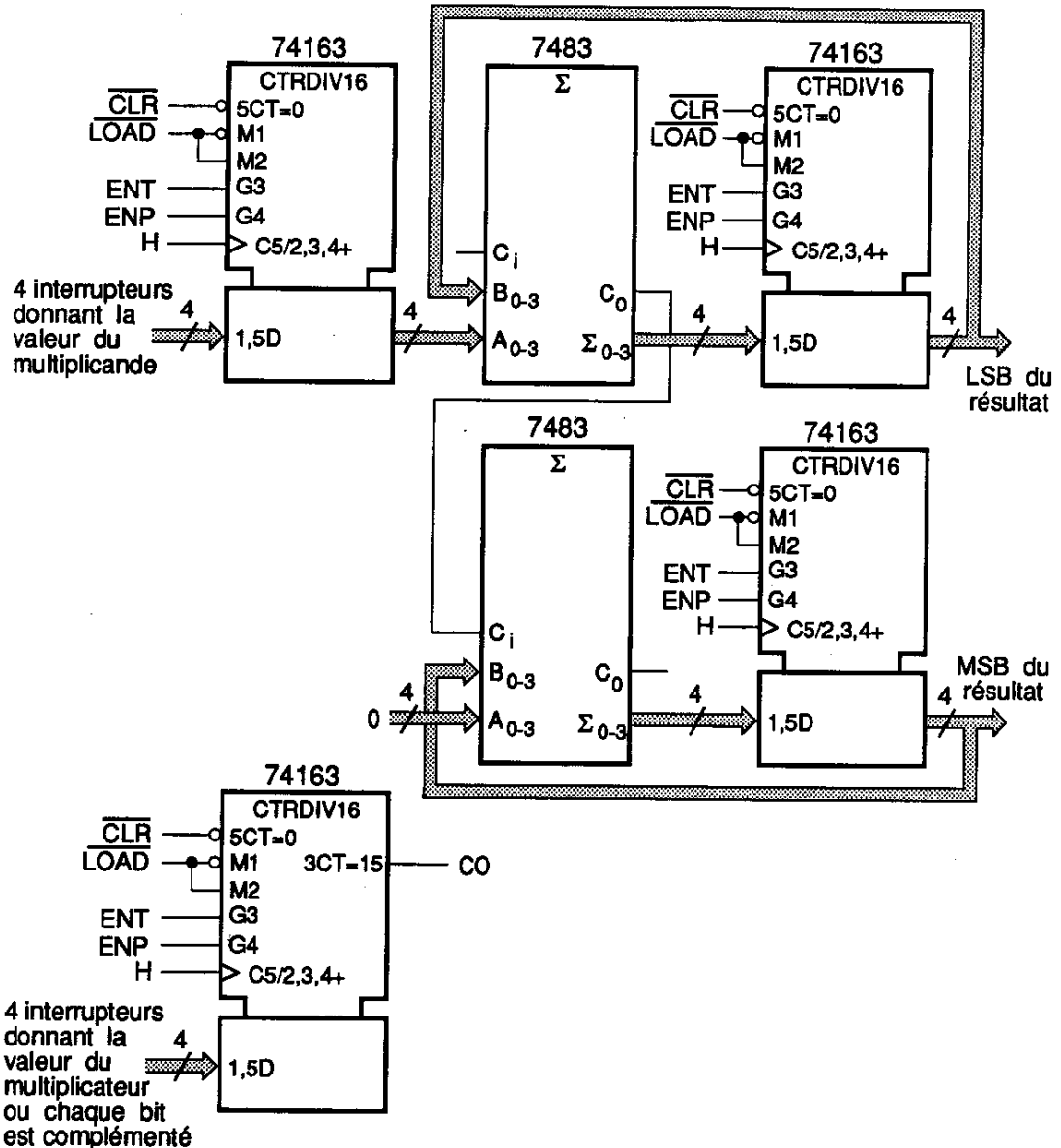


EFA : on parle dans l'organigramme de 3 régistres de mémorisation :

- MANDE [ ] qui sert qu'à préserver le multiplicande (4 bits)
- MATEUR [ ] qui emmagasine la valeur en complément restreint du multiplicateur. Il doit être capable de s'incrémenter (4 bits)
- RES [ ] qui contient le résultat partiel pendant le processus et le résultat valable à la fin. (8 bits ← multiplication 4 bits x 4 bits donne un résultat sur 8 bits)

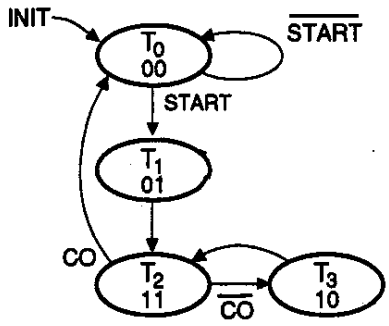
De plus, il est question de faire une addition

- additionneur 8 bits



EX: multiplicateur = 6 → (0110)
on change alors (1001)

Séquenceur :



codage : $Q_A\ Q_B$ (LSB)

tables des commandes J_i et K_i

J_A	Q_A	
	0	X
Q_B	1	X

$J_A = Q_B$

K_A	Q_A	
	CO	CO
Q_B	X	0
	CO	CO
Q_B	X	CO

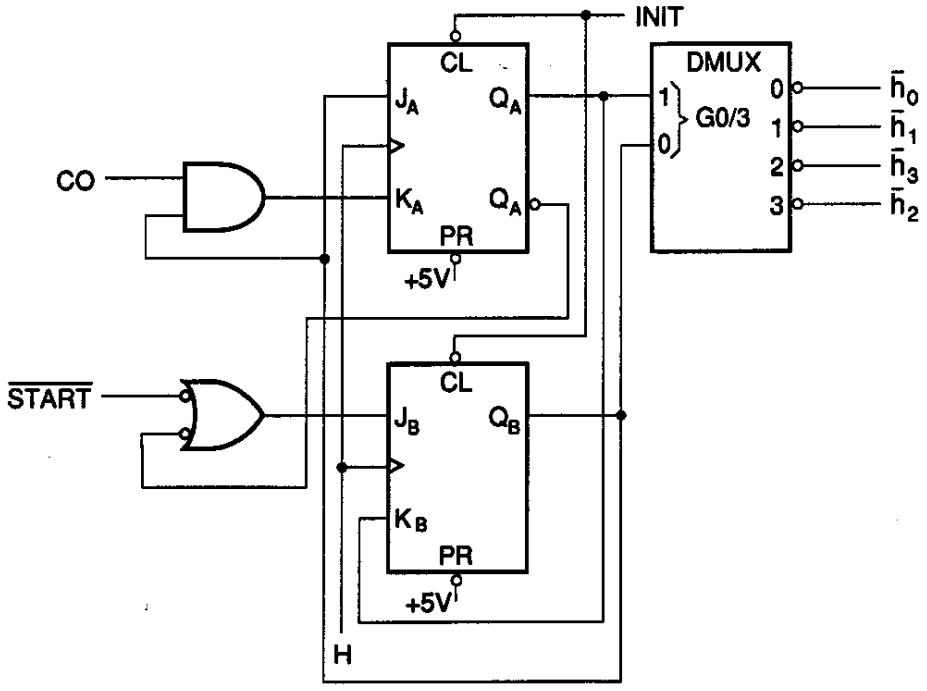
$K_A = CO \cdot Q_B$

J_B	Q_A	
	START	
Q_B	START	1
Q_B	X	X

$J_B = START + Q_A$

K_B	Q_A	
	X	X
Q_B	0	1

$K_B = Q_A$



Le séquenceur produit les phases (jointives ou disjointes) indiquant l'état présent dépendamment du codage choisi.

Ici : $h_0 \rightarrow T_0$, $h_1 \rightarrow T_1$, $h_2 \rightarrow T_3$ (ATTENTION!) , $h_3 \rightarrow T_2$ (ATTENTION!)

condition d'activation des commandes externes des EFA

→ régistre MANDE
chargement obligatoire dans l'état $T_1 \Rightarrow \bar{h}_1 = \overline{LOAD}$

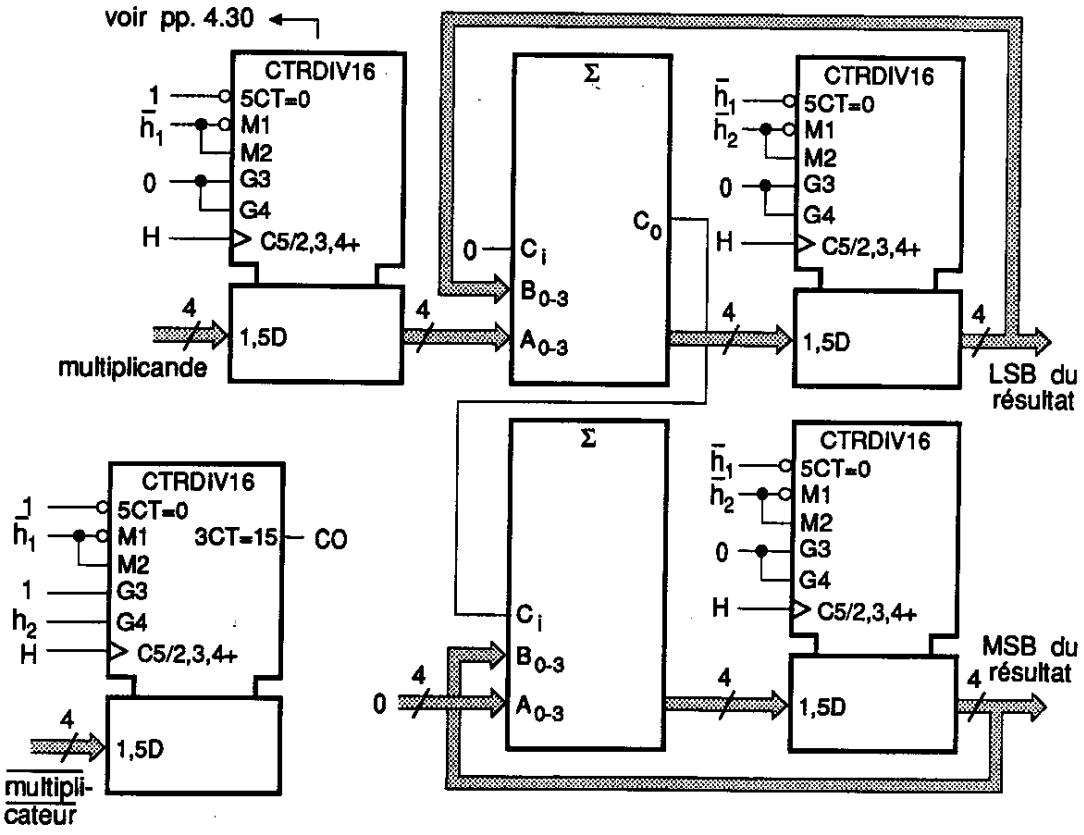
→ régistre MATEUR
chargement obligatoire dans l'état $T_1 \Rightarrow \bar{h}_1 = \overline{LOAD}$
incrément obligatoire dans l'état $T_3 \Rightarrow h_2 = ENP$
$ENT = 1$

→ régistre RES
mise à zéro obligatoire dans l'état $T_1 \Rightarrow \bar{h}_1 = \overline{CLR}$
chargement obligatoire dans l'état $T_3 \Rightarrow \bar{h}_2 = \overline{LOAD}$

→ sommateur
pas de retenue antérieure $\Rightarrow C_i = 0$

production des signaux de sortie

$PRET = h_0$
$TERMINE = h_3 \cdot CO$
↑
└ conditionnel à CO



8.5.2 - Multiplication par décalage et somme.

La multiplication par décalage et somme suit l'algorithme que nous employons pour multiplier sur papier.

Exemple :

```
  1 0 1 1
  0 1 1 0
  0 0 0 0
 1 0 1 1      <-- remarquez les décalages
1 0 1 1
0 0 0 0
1 0 0 0 0 1 0   <-- remarquez la somme
```

apportons une légère modification au précédé

```
  1 0 1 1
  0 1 1 0
  0 0 0 0
  0 0 0 0 :        décalage à droite
somme { 0 0 0 : 0
      { 1 0 1 1 :
  1 0 1 1 0        décalage à droite
somme { 1 0 1 : 1 0
      { 1 0 1 1 :
1 0 0 0 0 : 1 0    décalage à droite
somme { 1 0 0 0 : 0 1 0
      { 0 0 0 0 :
  1 0 0 0 0 1 0    décalage à droite
  0 1 0 0 0 0 1 0
```

résultat sur 2n bits

on peut éviter des sommations inutiles en demandant uniquement un décalage à droite lorsque le bit du multiplicateur examiné est "0".

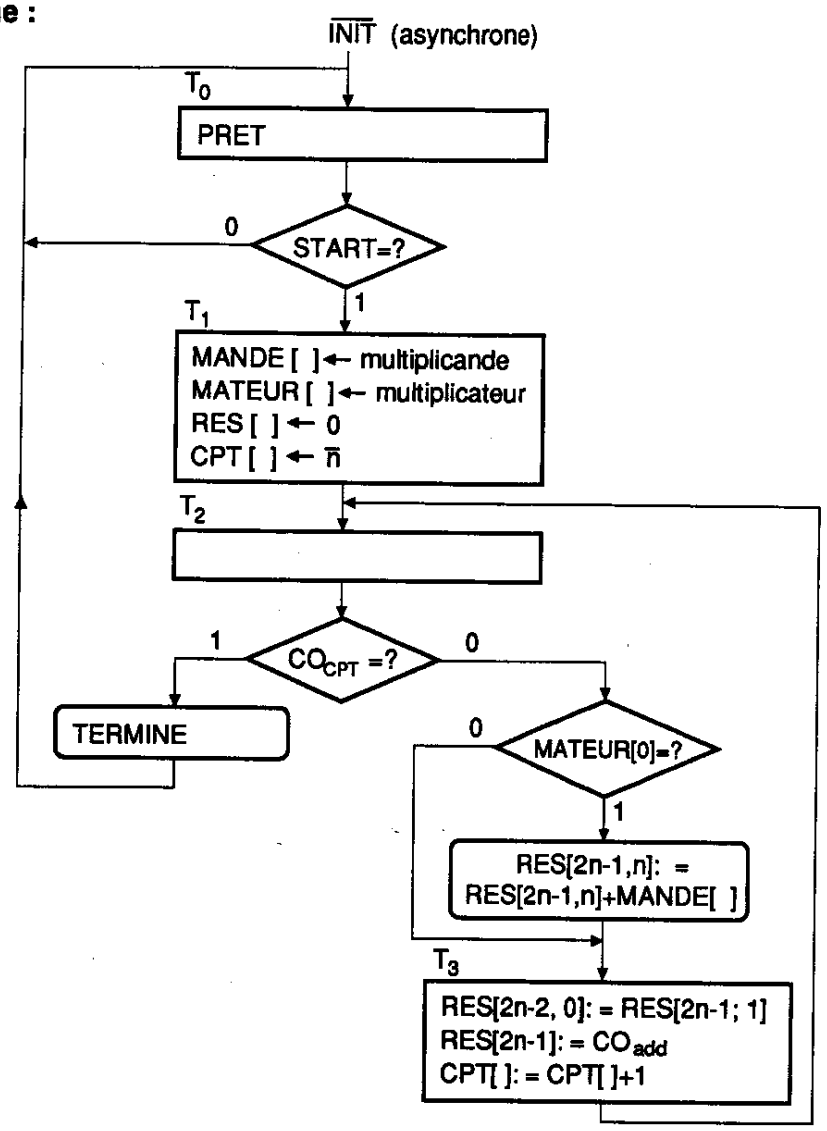
Algorithme :

```
programme muladdshift
    begin
    res: = 0;
    obtient (mande, mateur);   { n = nombre de bits des opérantes des produits
    for i: = 0 to (n-1) do
        begin
        flag: = mateur [bit i];
        if flag <> 0 then res [MSB]: = res [MSB] + mande;
        décalage (res, CO_ADD)
```

i.e. la retenue ← (* décalage à droite en rentrant CO_{ADD} de res *)

```
        (* res est constitué de 2n bits divisées en 2 groupes :
        le groupe des MSB et celui des LSB *)
            end;
    affiche (res)
    end.
```

Organigramme :



EFA : → MANDE [ ] : registre de mémorisation du multiplicande (4 bits)

→ MATEUR [ ] : registre de mémorisation à décalage du multiplicateur (4 bits)

→ RES [ ] : registre qui contient le résultat partiel pendant le processus et le résultat valable à la fin. Il doit être à décalage (8 bits)

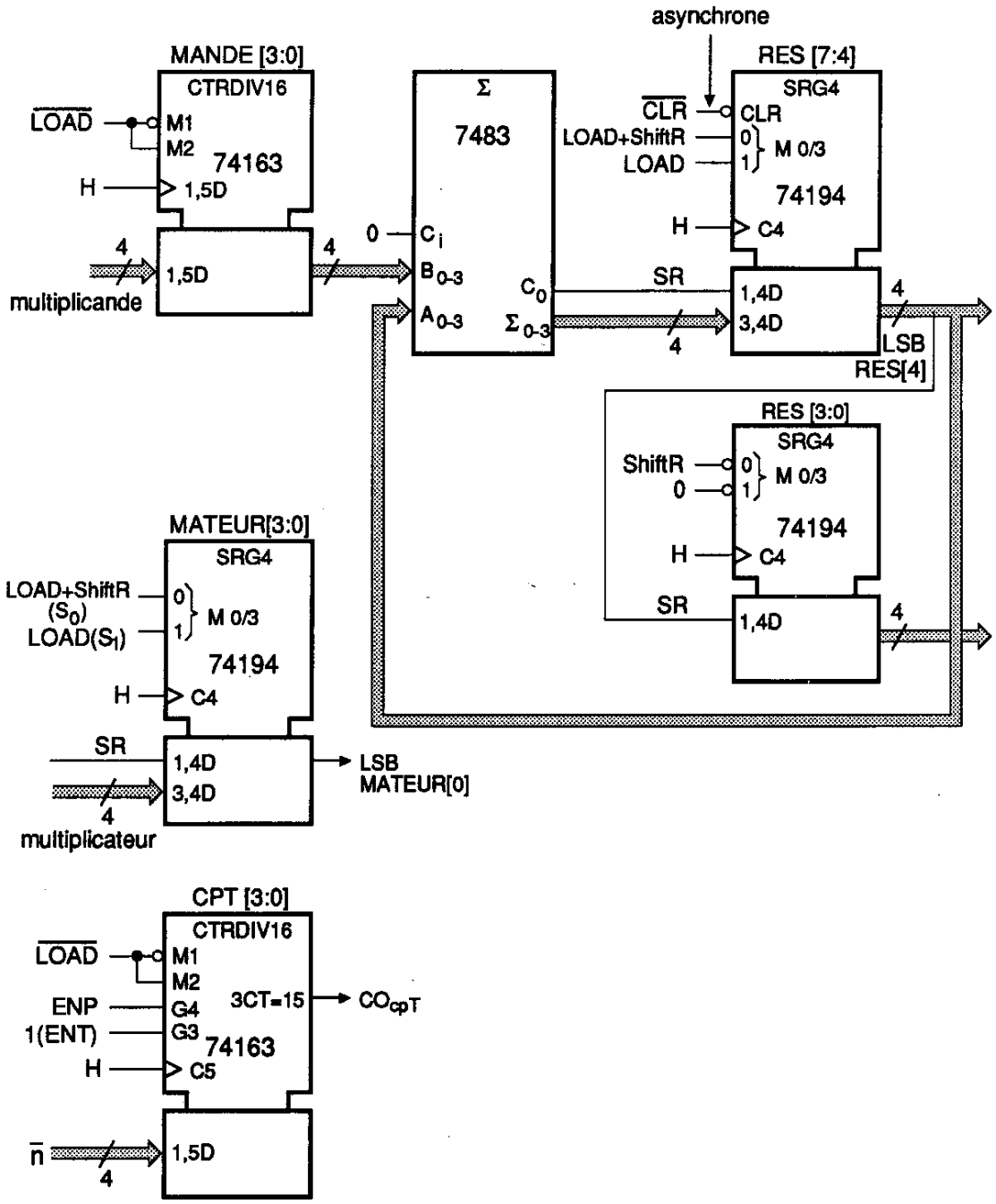
→ CPT [ ] : registre compteur indiquant si le nombre d'itérations est atteint (pour n=4, 2 bits suffiraient mais on prend un registre 4 bits)

→ additionneur 4 bits

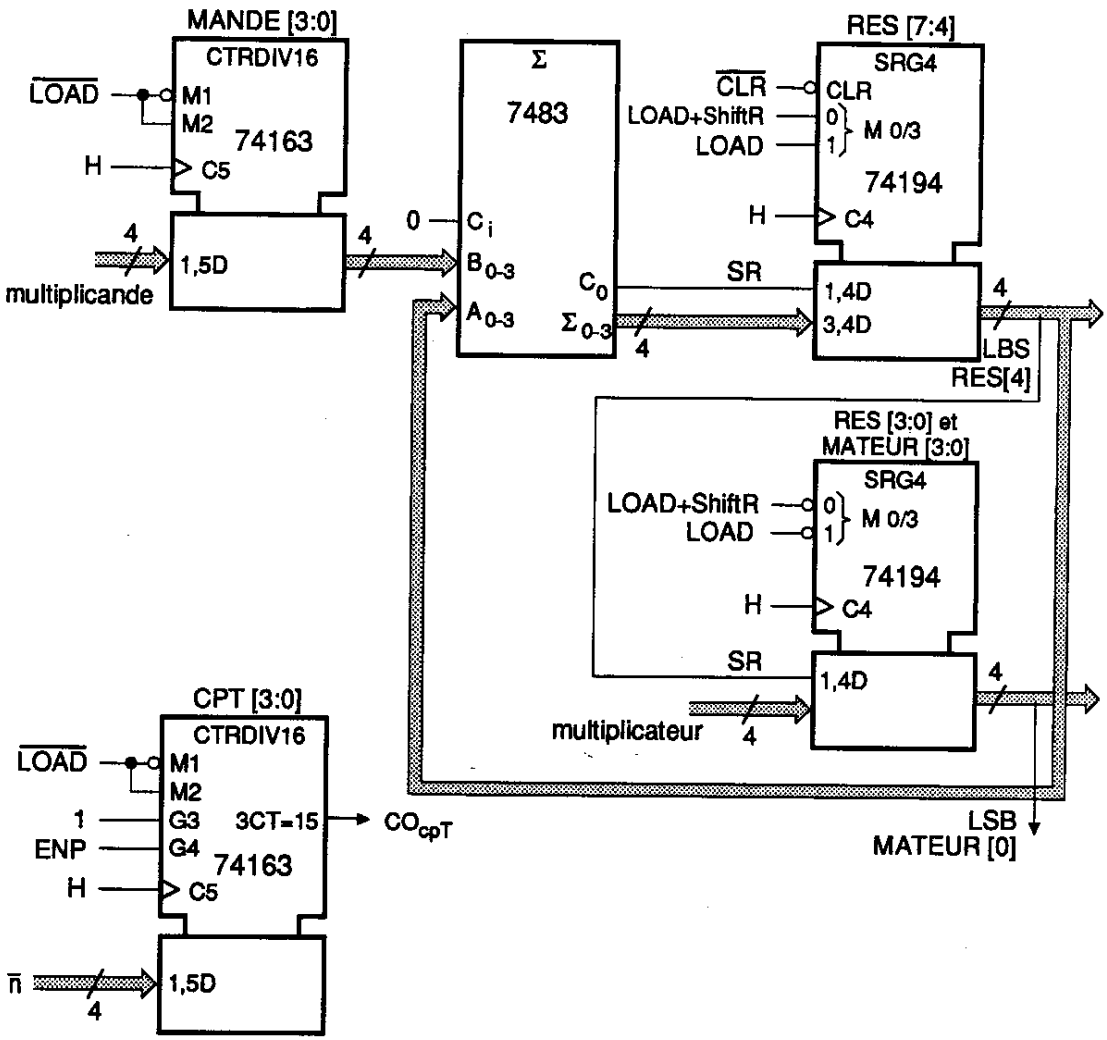
Note : on peut, en fait se servir d'un registre mémorisant à la fois le multiplicateur (au début) et les LSB du résultat (à la fin) car le décalage vers la droite du multiplicateur libère des bits utilisés pour le décalage à droite des MSB du résultat.

Le schéma bloc des interconnexions des EFA montre ceux-ci avec les commandes externes utiles seulement dans un but de simplification. Les autres commandes externes doivent être convenablement branchées à "0" ou à "1".

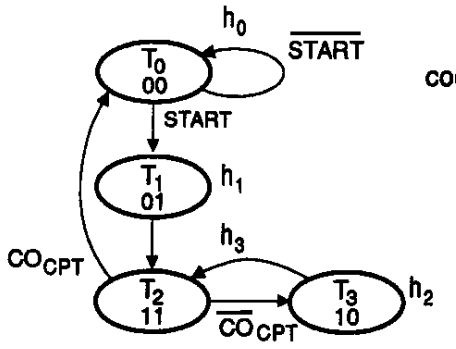
VERSION 1



VERSION 2 (préférable)



Séquenceur :



codage : $Q_A\ Q_B$ (LSB)

c'est le même que celui de la multiplication par additions successives bien que l'algorithme soit différent. (cf. séquenceur du paragraphe 8.5.1).

condition d'activation des commandes externes des EFA

→ régistre MANDE : $\overline{LOAD} = \bar{h}_1$ (compteur)

→ régistre MATEUR (RES [3;0]) : $S_0 = h_1 + h_2$
$S_1 = h_1$

→ régistre RES [7;4] : $\overline{CLR} = \bar{\Delta}_{h1}$ ← phase disjointe obligatoire (sans aléa) $\bar{H} \cdot h_1$
$S_0 = h_3 \cdot MATEUR\,[0] \cdot \overline{CO}_{CPT} + h_2$
$S_1 = h_3$

on remarque que l'enregistrement de la sommation s'effectue en T_2 en autant que MATEUR [0] soit égal à 1.

→ $Q_A = CO_{ADD}$

→ régistre CPT : $\overline{LOAD} = \bar{h}_1$
$ENP = h_2$

production des signaux de sortie

→ $PRET = h_0$

→ $TERMINE = h_3 \cdot CO_{CPT}$

attention : Les phases jointives peuvent contenir des aléas. Elles ne doivent pas être appliquées sur des entrées asynchrones.

8.5.3. - Multiplication en complément 2 (algorithme de Booth).

Les opérations arithmétiques composent l'ensemble des opérations les plus utilisées dans les circuits numériques le traitement (micro-processeur, processeur câblé moyen (VAX-785), ordinateur...). La multiplication, entre autre, consomme à elle seule un temps de calcul énorme atteingnant 20 fois celui d'une addition sur un même nombre de bits (multiplication et addition entières en complément 2).

Il convient donc d'utiliser un algorithme approprié pour réaliser une multiplication de nombres codés complément 2 puisque c'est de cette façon, rappelons-le, qu'un nombre est représenté dans la mémoire d'un ordinateur. Cet algorithme est celui de Booth.

Reprenons la représentation des nombres en complément 2 (cf paragraphe 1.5.3). Une brève analyse permet de reconfigurer le poids des bits selon leur position.

$$(N)_{2,CV} = (a_{n-1}\ a_{n-2} \ldots a_1\ a_0)$$

valeur de $N = a_{n-1}(-2^{n-1}) + a_{n-2}(2^{n-2}) + \ldots a_1(2^1) + a_0(2^0)$

$$a_j(-2^j) + a_{j-1}\,2^{j-1} = (a_{j-1} - a_j)2^j - a_{j-1}\,2^{j-1}$$ poids des bits selon leur position

valeur de $N = 2^{n-1}(a_{n-2} - a_{n-1}) + 2^{n-2}(a_{n-3} - a_{n-2}) + \ldots + 2^1(a_0 - a_1) + 2^0(0 - a_0)$

Exemple : $(N)_{2,CV} = (1011)$

$(N)_{10} = 1(-2^3) + 1(2^1) + 1(2^0)$

$= -8+2+1 = -5$

ou

$(N)_{10} = 2^3(0-1) + 2^2(1-0) + 2^1(1-1) + 2^0(0-1)$

$= -8+4-1 = -5$

Cette dernière expression de la valeur selon le poids s'apparente d'avantage à celle lorsqu'aucun codage i.e.:

$$(N)_2 = (a_{n-1}\ a_{n-2} \ldots a_1\ a_0)$$

$$(N)_{10} = 2^{n-1}(a_{n-1}) + 2^{n-2}(a_{n-2}) + \ldots 2^1(a_1) + 2^0(a_0)$$

On peut donc calquer l'algorithme de l'addition par décalage et somme, valide sans codage, pour le rendre valide avec un codage en complément 2 : examiner tour à tour une des bits du multiplicateur; si la bit est "1", additionner le multiplicande au résultat partiel; décaler vers la droite le résultat partiel. Le calquage est rendu possible à cause de la décroissance réalisée par le décalage à droite.

$$(N)_{10} = 2^{n-1}(\ ) + 2^{n-2}(\ ) + \ldots 2^1(\ ) + 2^0(\ )$$

La différence se situe au niveau du terme à l'intérieur des parenthèses.

- $(N)_2$ → terme associé au poids $2^x = a_x$
- $(N)_{2,CV}$ → terme associé au poids $2^x = a_{x-1} - a_x$

Selon la valeur du terme, plusieurs actions peuvent être entreprises. Évidemment, dans la cas ou nous avons $(N)_2$, le terme ne peut prendre que les valeurs "1" ou "0" qui conduisent respectivement vers les actions "additionne le multiplicande et décale", "décale seulement". Lorsque N est codé complément 2 ($(N)_{2,CV}$), le terme peut prendre une nouvelle valeur : "-1". L'action entreprise sera alors "soustrait le multiplicande et décale".

0 → décale
1 → additionne multiplicande, décale
− 1 → soustrait multiplicande, décale

L'algorithme de Booth utilise cette astuce, laquelle se réalise par le biais de comparaisons de 2 bits successives du multiplicateur: (MATEUR [x], MATEUR [x-1]).

MATEUR[x]	MATEUR[x-1]	actions
0	0	décalage du résultat partiel
0	1	addition du multiplicande au résultat partiel, décalage
1	0	soustraction du multiplicande, décalage
1	1	décalage

* les opérations d'addition et de soustraction sont réalisées en complément 2.

Algorithme :

```
programme Booth
    begin
    res: = 0;
    flag [old]: = 0;
    obtient (mande, mateur);
    for i: = 0 to (n-1) do
        begin
        flag [new]: = mateur [bit i];
(* ainsi flag [old] correspond à MATEUR [x-1]
  et flag [new], à MATEUR [x] *)
        if flag [old] <> flag [new] then
            if flag [new] = 1 then
                res [MSB]: = res [MSB] - mande
            else res [MSB]: = res [MSB] + mande;
        décalage (res, res [bit 2n-1]);
(* décalage à droite de la valeur res [bit 2n-1] de res *)
(* res est constitué de 2n bits divisées en 2 groupes
  le groupe des MSB et celui des LSB *)
        flag [old]: = flag [new]
        end;
    affiche (res)
    end
```

Notes :

- le décalage à droite de la valeur res [bit 2n-1] de res permet de faire l'extension du signe.

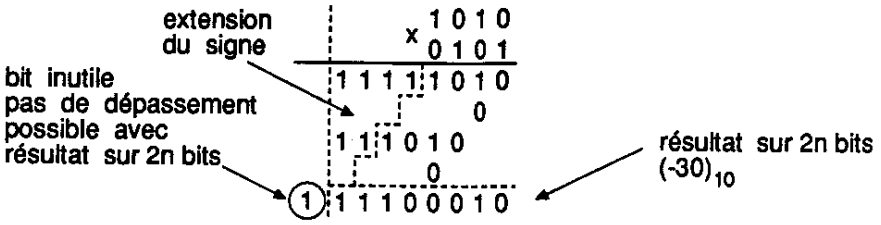
  L'extension du signe conduit vers une autre solution possible de la multiplication en complément 2. Mais cette version est plus lente et ne fonctionne que si le multiplicateur est supérieur à 0.

  exemple : $(-3)_{10} = (101)_{2,CV}$ sur 3 bits
  $= (1101)_{2,CV}$ sur 4 bits
  etc...

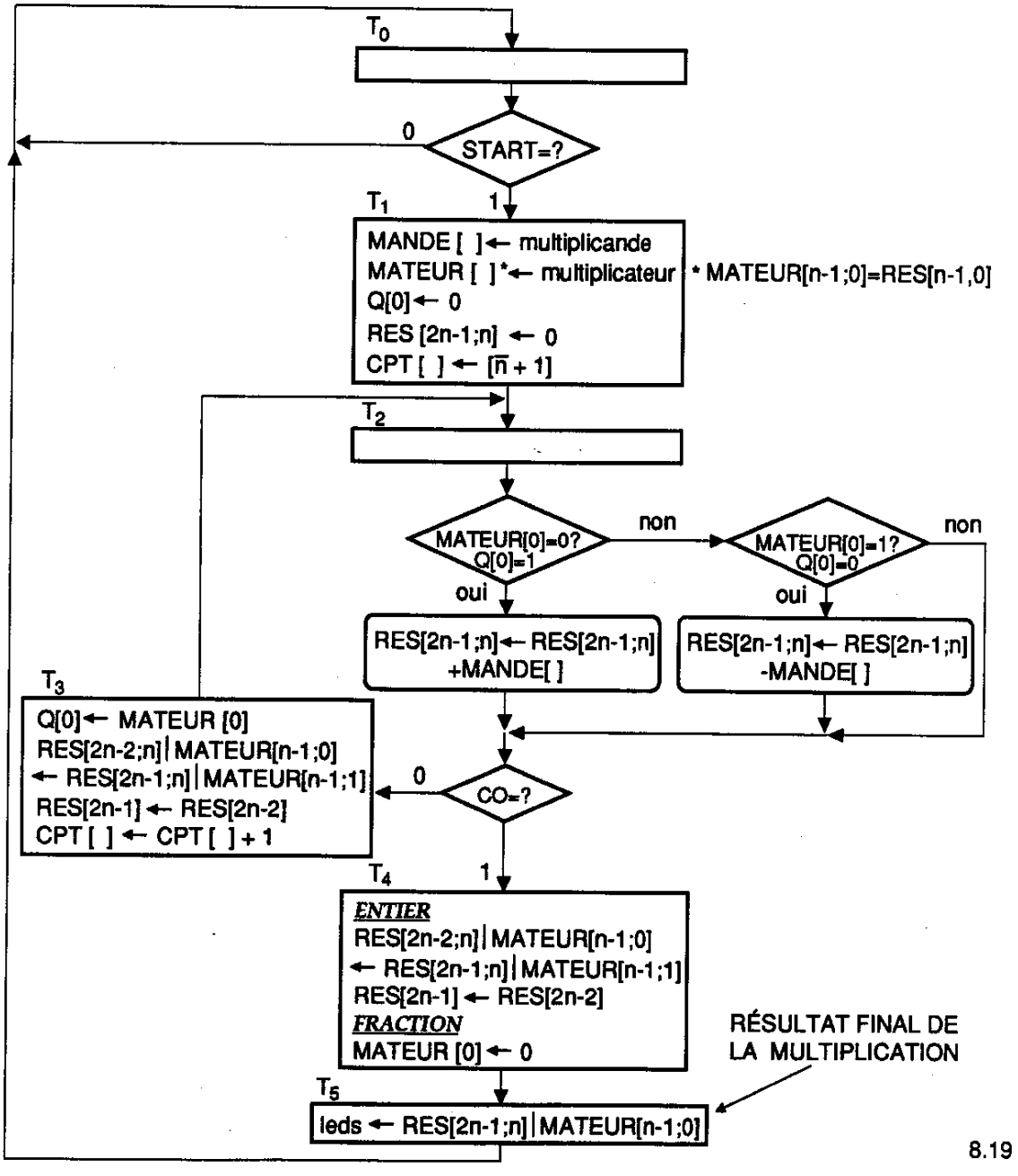
- on peut modifier légèrement l'algorithme pour le rendre opérationnel avec des nombres fractionnaires, il s'agit d'éliminer le dernier décalage demandé et de forcer res [bit 0] = 0.
- on peut utiliser un seul et même régistre pour mémoriser le multiplicateur initialement et les LSB du résultat finalement. Le tout suivant le principe démontré lors de la réalisation du multiplicateur par addition et décalage. Il faut cependant ajouter un régistre d'un bit conservant ce que l'algorithme appelait flag [old].

Exemple : multiplicande = $(-6)_{10} = (1010)_{2,CV}$

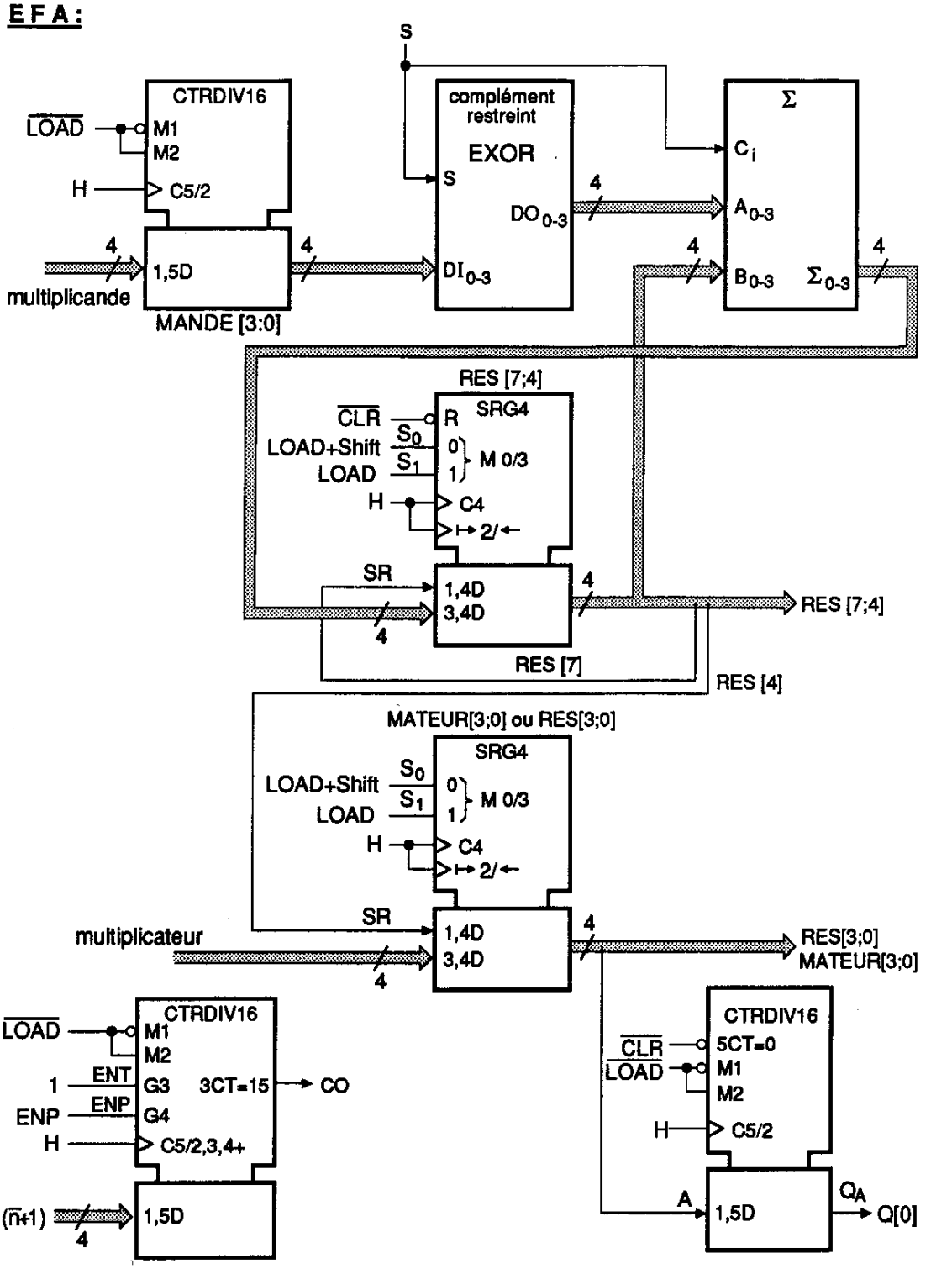
multiplicateur = $(5)_{10} = (0101)_{2,CV}$



Organigramme :



E F A :



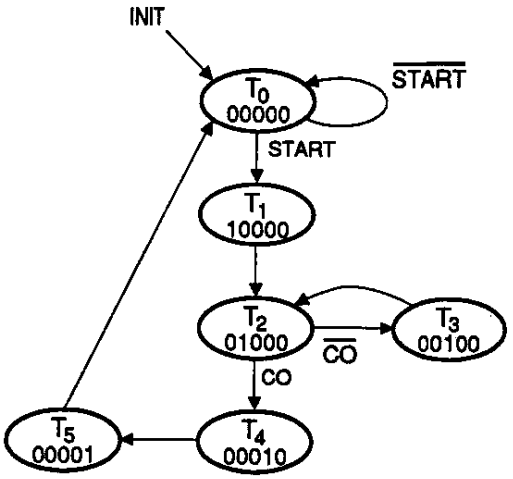
NOTE : Par souci de simplification, seuls les commandes utiles (exigées par l'organigramme) apparaissent. Les autres doivent être branchées convenablement pour le bon fonctionnement (ex: $\overline{CLR}$ de MANDE[3;0]=1).

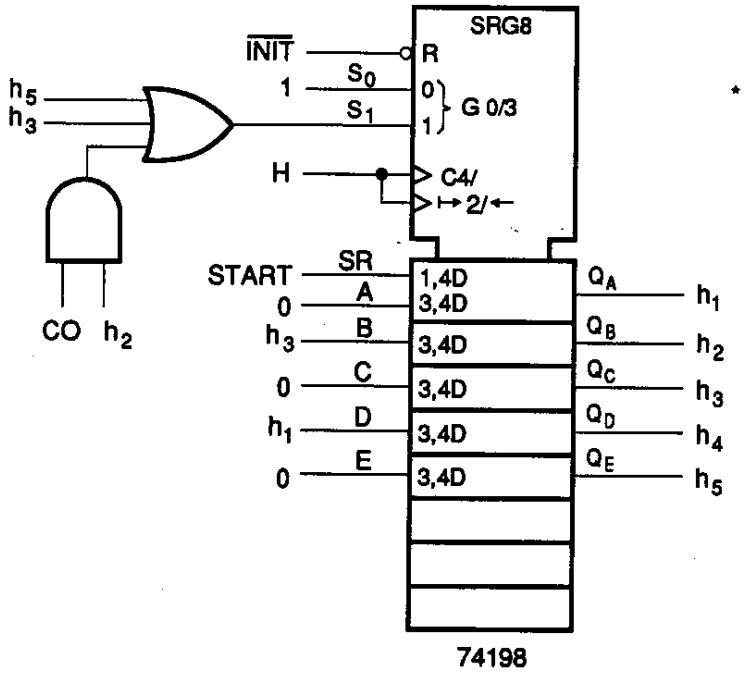
séquenceur :

codage :

$Q_A Q_B Q_C Q_D Q_E$

LSB





* START n'arrive que pendant T_0

$$h_0 = \overline{h_1 + h_2 + h_3 + h_4 + h_5} = \bar{h}_1 \cdot \bar{h}_2 \cdot \bar{h}_3 \cdot \bar{h}_4 \cdot \bar{h}_5$$

mais

on a pas besoin de h_0 car aucune action sur les EFA ou sur des signaux de sortie n'est exécutée pendant la phase T_0.

Conditions d'activation des commandes externes des EFA

- registre MANDE : $\overline{LOAD} = \bar{h}_1$
- registre MATEUR (RES [3;0]) : $S_0 = h_1 + h_3 + h_4$
  $S_1 = h_1$
- registre RES [7;4] : $\overline{CLR} = \Delta\bar{h}_1$
  $S_0 = h_2$ (MATEUR [0] $\oplus$ Q[0]) + $h_3 + h_4$
  $S_1 = h_2$ (MATEUR [0] $\oplus$ Q[0])

  l'enregistrement de la sommation (ou différence) s'effectue en T_2 à la condition que mateur [0] (ou flag [new]) soit différent de Q[0] (ou flag [old]). Ceci explique le "ou exclusif"

- registre Q[0] : $\overline{CLR} = \bar{h}_1$
  $\overline{LOAD} = \bar{h}_3$
- registre CPT : $\overline{LOAD} = \bar{h}_1$
  EN P $= h_3$
- circuit du complément restreint : $S = \overline{Q[0]}$

  la soustraction est demandée en T_2 lorsque MATEUR [0] = 1 et Q[0] = 0. Mais puisque l'enregistrement du résultat se fait en T_2 lorsque MATEUR [0] sont différents, on peut simplifier la demande de soustraction à $\overline{Q[0]}$.

> **Note :** la soustraction en complément 2 se réalise simplement en additionnant les opérandes en complémentant préalablement le second. (cf. annexe C).

Exemple de l'algorithme de Booth :

2 nombres de 4 bits ($n = 4 = (0100)_2$)
multicande = $(1010)_{2,CV} = (-6)_{10}$
multiplicateur = $(1101)_{2,CV} = (-3)_{10}$

MANDE [3;0] = (1010)
MATEUR [3;0] = (1101)
Q[0] = (0)
RES [7;4] = (0000)
CPT [3;0] = $(\bar{0}\bar{1}\bar{0}\bar{0}) + 1 = (1100)$

(MATEUR [0] = 0 ?) NON → (MATEUR [0] = 1 ?) OUI
(Q [0] = 1) (Q [0] = 0)

RES [7;4] = $(0000)_{2,CV} - (1010)_{2,CV}$
RES [7;4] = $(0110)_{2,CV}$

(CO = 1 ?) NON
Q[0] = (1)
RES [6;4] | MATEUR [3;0] = (0110110)
RES [7] = 0
donc MATEUR [3;0] = (0110) RES [7;4] = (0011)
CPT [3;0] = (1101)

(MATEUR [0] = 0 ?) OUI
(Q [0] = 1)

RES [7;4] = $(0011)_{2,CV} + (1010)_{2,CV}$
RES [7;4] = $(1101)_{2,CV}$

(CO = 1 ?) NON
Q[0] = 0
RES [6;4] | MATEUR [3;0] = (1101011)
RES [7] = 1
donc MATEUR [3;0] = (1011) RES [7;4] = (1110)
CPT [3;0] = (1110)

(MATEUR [0] = 0 ?) NON → (MATEUR [0] = 1 ?) OUI
(Q [0] = 1) (Q [0] = 0)

RES [7;4] = $(1110)_{2,CV} - (1010)_{2,CV}$
RES [7;4] = $(1110)_{2,CV} + (0110)_{2,CV}$
RES [7;4] = $(0100)_{2,CV}$

(CO = 1 ?) NON
Q[0] = 1
RES [6;4] | MATEUR [3;0] = (0100101)
RES [7] = 0
donc MATEUR [3;0] = (0101) RES [7;4] = (0010)
CPT [3;0] = (1111)

(MATEUR [0] = 0 ?) NON → (MATEUR [0] = 1 ?) NON
(Q [0] = 1) (Q [0] = 0)

(CO = 1 ?) OUI

entier ⟶ RES [6;4] | MATEUR [3;0] = (0010010)

RES [7] = 0

donc RES [7;0] = $(0001,0010)_{2,CV} = (18)_{10}$

* Si on suppose que les nombres sont fractionnels alors on a :

multiplicande = $(1.010)_{2,CV} = (-3/4)_{10}$

multiplicateur = $(1.101)_{2,CV} = (-3/8)_{10}$

la dernière étape devient dans ce cas

fraction ⇢ MATEUR [0] = 0

donc REST [7;0] = $(0.0100100)_{2,CV} = (9/32)$

8.54 - distributrice à café

Vous faites partie de l'une des 2 équipes chargées de réaliser une machine à café avec séquenceur électronique.

Votre équipe est responsable de la conception de la partie centrale électronique, tandis que l'autre équipe se concentre sur la réalisation des 3 modules périphériques suivants :

- DG 101 : circuit qui trie la monnaie par le biais de leur grosseur et qui évacue l'ensemble des pièces insérées sur la commande TTL "Évacue Pièces" (EP). Le circuit fournit 5 signaux synchrones TTL :
  * "pièce présente" (PP)
  * "pièce de 0.50$ insérée" (50I)
  * "pièce de 0.25$ insérée" (25I)
  * "pièce de 0.10$ insérée" (10I)
  * "pièce de 0.05$ insérée" (05I)

- DG 201 : circuit qui distribue par coup de 0.05$. Le circuit répond à la commande "retourne 0.05$" (R05) et envoie le signal synchrone "change remis"(CR) lorsque la distribution est effectuée.

- DG 301 : circuit qui sert le café. Le circuit fait couler le café tant et aussi longtemps que la commande "sert café" (SC) est activée. Cependant, lorsque la quantité nécessaire est atteinte, il envoie un signal synchrone "café prêt" (CP).

Ces 3 modules deviennent donc des EFA de votre système.

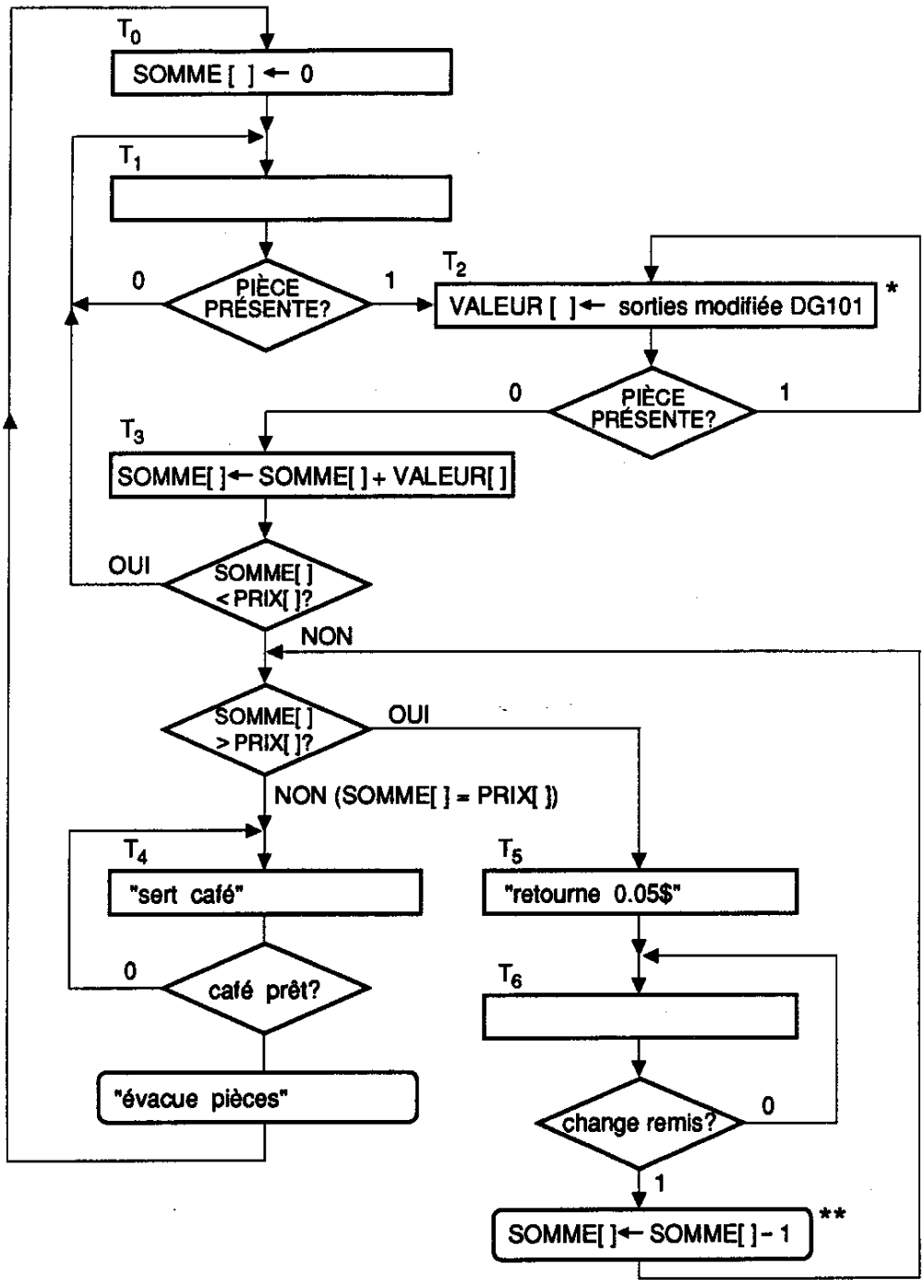
Votre chef d'équipe, après une longue expertise, a développé l'algorithme puis l'organigramme ci-dessous. Il vous suggère aussi d'enregistrer des valeurs binaires qui correspondent aux valeurs réelles divisées par 5 puisque la machine n'accepte pas les pièces de 0.01$.

Algorithme : programme café

```
begin
somme : = 0;
repeat

    wait (pièce présente);
(*attend que le signal TTL "pièce présente" soit actif*)
    repeat (détermine valeur)
        until not (pièce présente);
(*détermine la valeur de la pièce insérée*)
    somme : = somme + valeur
until (somme ≥ prix);
while (somme > prix) then
    begin
    change ;
(*retourne 0.05$ de change*)
    somme : = somme - 0.05$;
    end;

    repeat (sert) until (café prêt);
    évacue
(*sert le café jusqu'à la quantité requise*)

    end.
```

Votre chef vous stipule que la machine n'a pas à tenir compte d'une somme supérieure à 0.75$ et que la machine doit posséder un ensemble d'interrupteurs réglant le coût du café (entre 0.05$ et 0.75$ par tranche de 0.05$).

Organigramme :

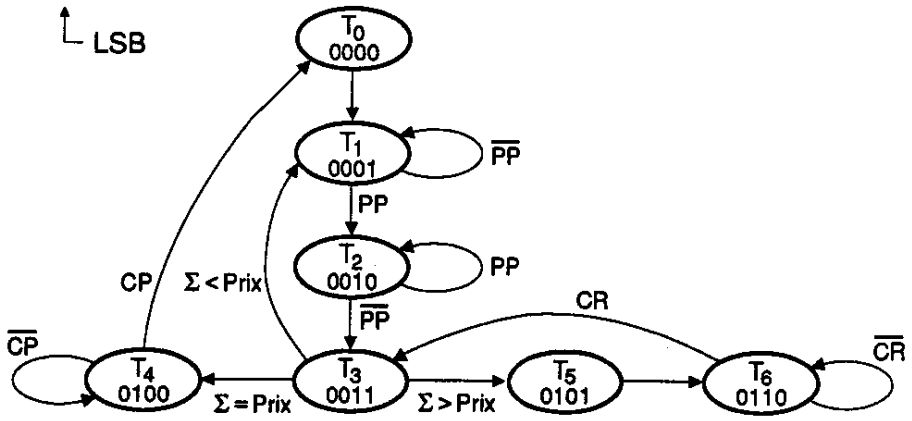


* les sorties modifiées de DG101 donnent en bits, la valeur divisée par 5 de la pièce insérée.

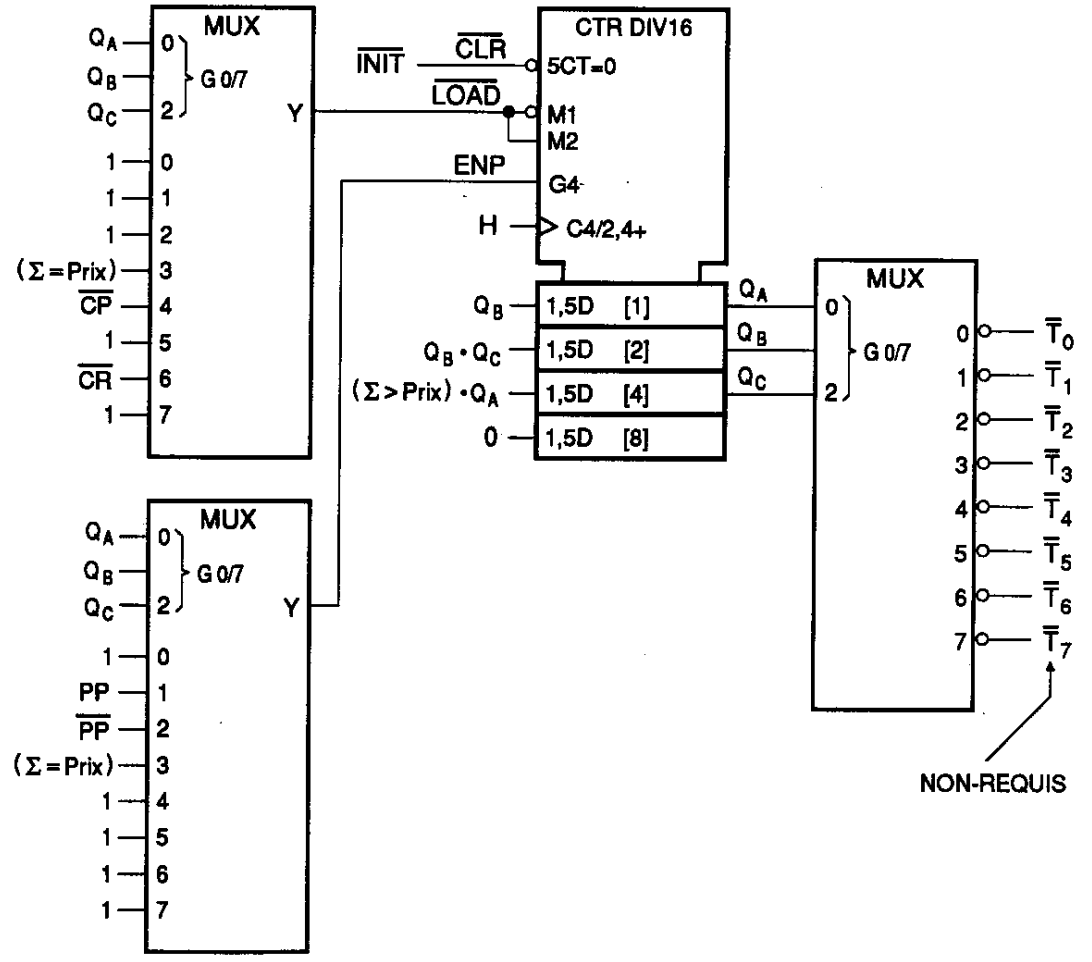
** SOMME [ ] ← SOMME [ ] - 1 correspond à soustrait 0.05$ de la somme en sous.

Graphe :

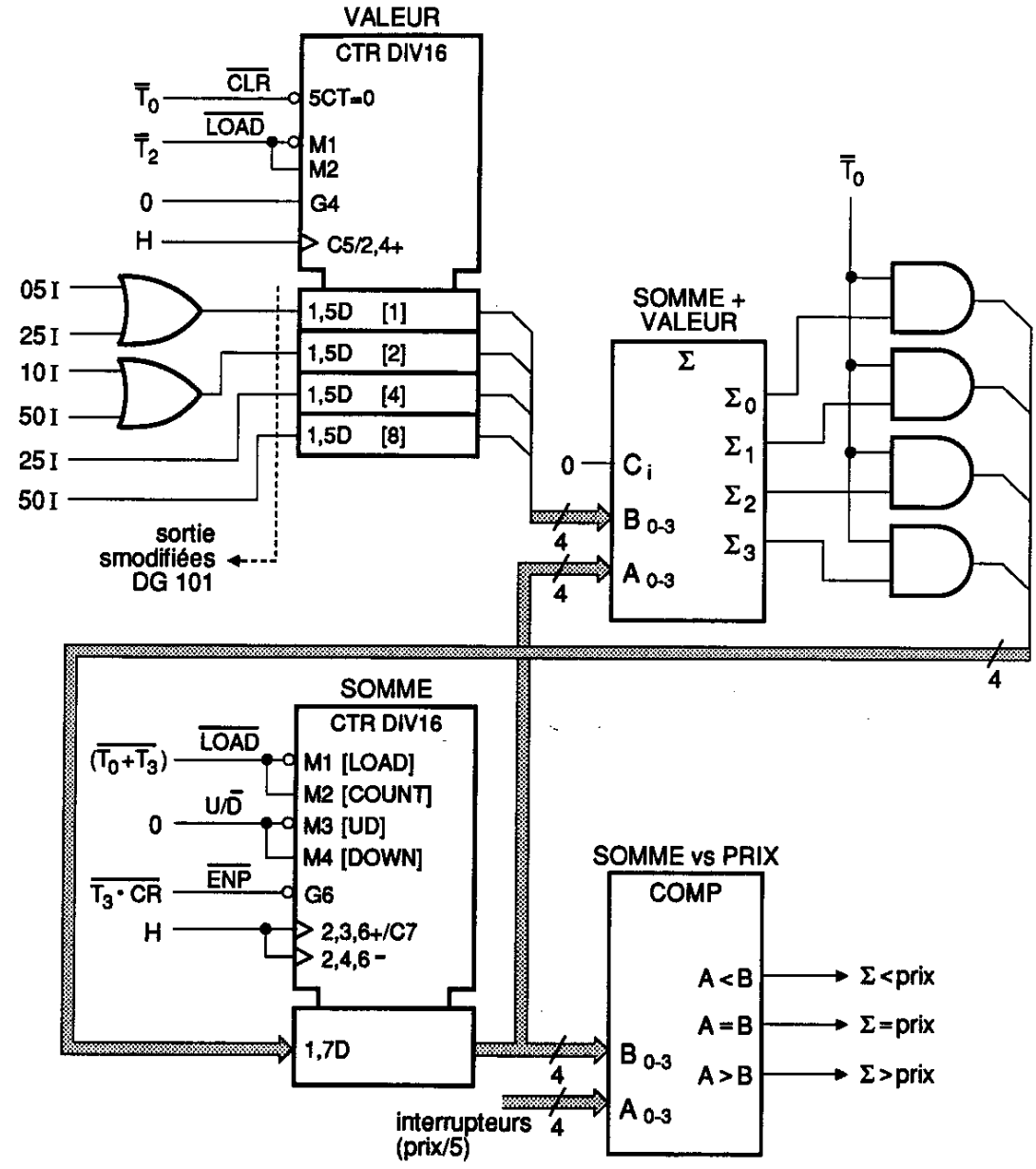
codage : $Q_D Q_C Q_B Q_A$

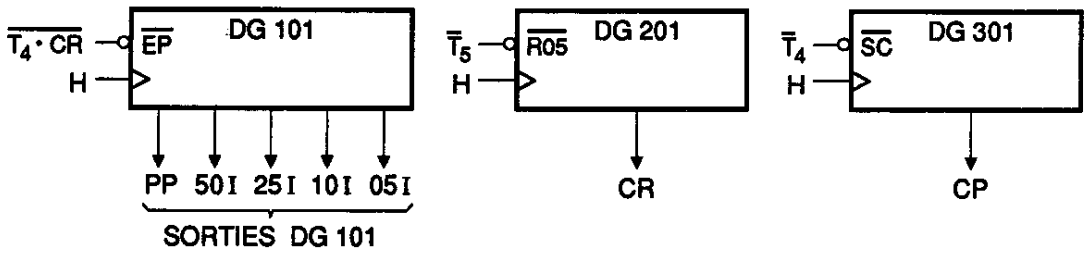


Séquenceur :



E F A :


VALEUR
CTR DIV16
$\overline{T_0}$
$\overline{CLR}$
5CT=0
$\overline{T_2}$
$\overline{LOAD}$
M1
M2
0
G4
H
C5/2,4+
05 I
25 I
10 I
50 I
25 I
50 I
1,5D [1]
1,5D [2]
1,5D [4]
1,5D [8]
sortie
smodifiées
DG 101
SOMME +
VALEUR
Σ
0
C_i
B_{0-3}
A_{0-3}
Σ_0
Σ_1
Σ_2
Σ_3
4
4
4
SOMME
CTR DIV16
$(\overline{T_0+T_3})$
$\overline{LOAD}$
M1 [LOAD]
M2 [COUNT]
0
U/$\overline{D}$
M3 [UD]
M4 [DOWN]
$\overline{T_3 \cdot CR}$
$\overline{ENP}$
G6
H
2,3,6+/C7
2,4,6 -
1,7D
SOMME vs PRIX
COMP
B_{0-3}
A_{0-3}
A<B
A=B
A>B
Σ<prix
Σ=prix
Σ>prix
4
4
interrupteurs
(prix/5)


$\overline{T_4 \cdot CR}$
H
$\overline{EP}$
DG 101
PP 50 I 25 I 10 I 05 I
SORTIES DG 101
$\overline{T_5}$
H
$\overline{R05}$
DG 201
CR
$\overline{T_4}$
H
$\overline{SC}$
DG 301
CP

Département de Génie électrique

GEL-10279 CIRCUITS LOGIQUES

NOTES DE COURS

CHAPITRE 9

CIRCUITS SEQUENTIELS ASYNCHRONES

Définition

Les circuits asynchrones peuvent être considérés comme les fondements des machines séquentielles : le coeur des circuits séquentiels synchrones, les bascules, est un circuit asynchrone. Ainsi, la logique voudrait que soit présentée la conception des circuits asynchrones avant celle des circuits synchrones mais les problèmes inhérents à l'asynchrone obligent l'inversion des présentations. Ces problèmes tels que les aléas dynamiques, statiques ou essentiels, les courses critiques et les oscillations doivent être compris et pris en considération. La méthode de design développée par Hoffman est la méthode reconnue et tient compte d'une bonne gamme de facteurs lorsque appliquée à bon escient.

Dans un circuit séquentiel synchrone, des éléments "mémoire" commandés par une horloge commune forment la base du principe avec rétroaction. Les sorties des éléments "mémoire" fournissent l'état de la machine, lequel est décodé pour former les sorties du circuits. Quant à l'évaluation de l'état suivant, elle dépend de l'état présent et des commandes d'entrée.

Dans un circuit séquentiel asynchrone, les éléments "mémoire" de type bascule synchrone sont remplacés par les délais de propagation des circuits combinatoires de décodage. Ce sont ces délais qui permettent la rétroaction nécessaire à un circuit séquentiel.

La figure 9.1 illustre le schéma général d'un circuit séquentiel asynchrone. On y note :

- l'absence des bascules synchrones
- l'apparence d'un circuit combinatoire avec rétroaction
- l'absence d'une commande de synchronisation
- le principe de base de la rétroaction : un circuit combinatoire idéal suivi d'éléments "mémoire" du type à délai.

La rétroaction sur n variables d'un circuit combinatoire peut, si le circuit fonctionne correctement, produire 2^n états stables. Mais cette rétroaction critique implique l'utilisation sévère d'une méthode de design, et la compréhension des multiples problèmes d'utilisation intuitive des circuits logiques qui se présentent lorsqu'on néglige les caractéristiques dynamiques des éléments.

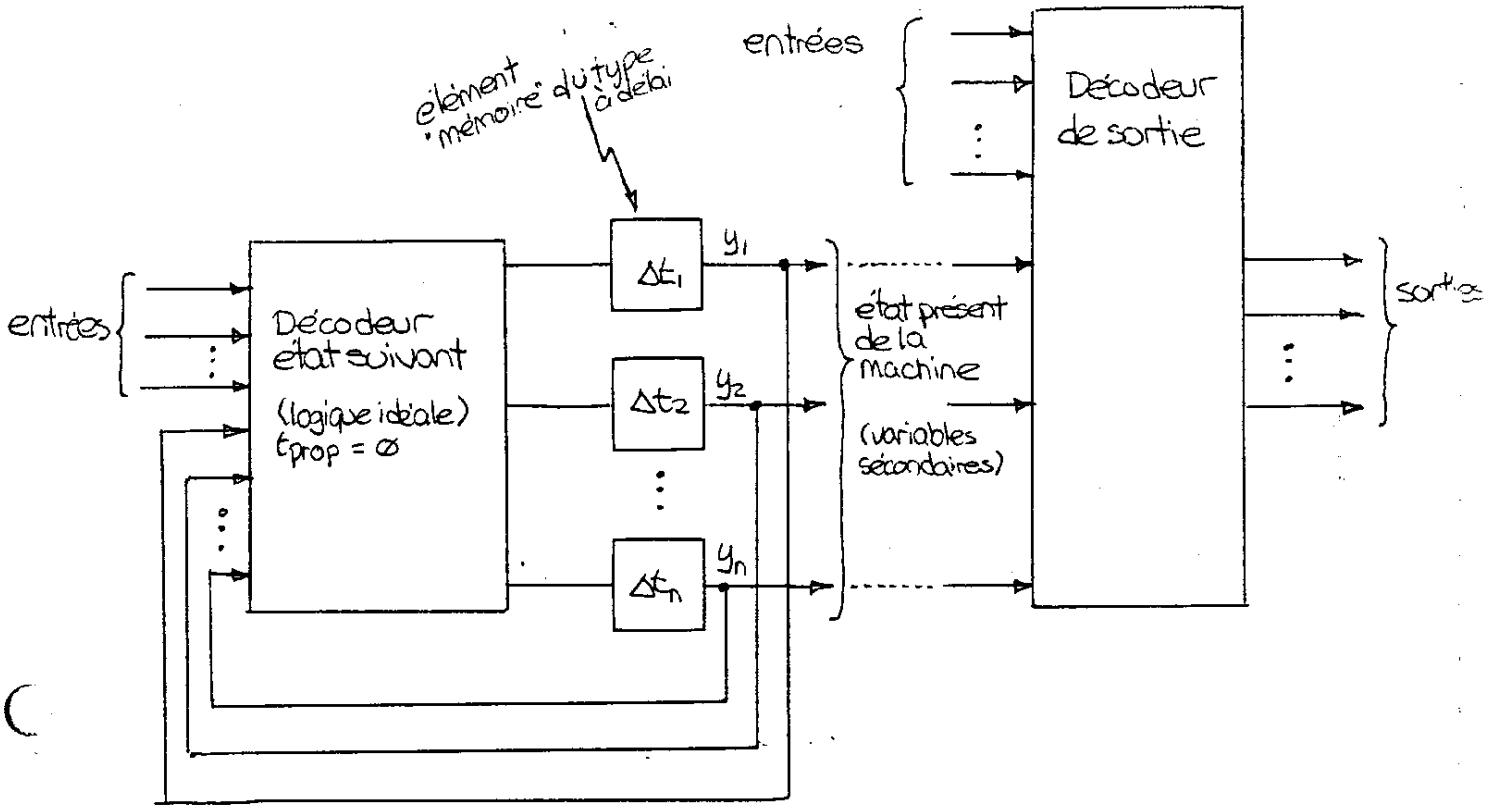


Figure 9.1 Schéma général d'un circuit séquentiel asynchrone

La conception d'un circuit séquentiel asynchrone vient de la nécessité d'un circuit avec séquences mais dont le circuit synchrone ne peut être réalisé pour les raisons suivantes :

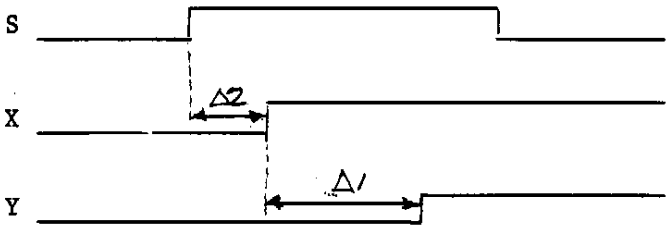
- l'horloge n'est pas disponible ou utilisable
- la vitesse requise dépasse la capacité d'un circuit synchrone

Exemple de base

Le circuit asynchrone de base le plus simple est la cellule binaire tel que représenté par la figure 9.2. Au point de vue statique, la rétroaction fournit l'équation implicite

$$X = S + \overline{R} \cdot X$$

C'est l'analyse dynamique, qui tient compte des délais de propagation des portes, qui montre certains comportements d'opération. Ainsi, toujours avec S = R = 0, initialement, on apprend que le passage de la commande S de 0 à 1 entraîne une mise à 1 forcée de la sortie X en autant que la commande dure plus longtemps que la somme des délais de propagation des deux portes.



avec

$$\Delta = \Delta_1 + \Delta_2$$

Au point de vue dynamique, il convient donc de réaménager l'équation de la cellule binaire et de tenir compte des délais localisés à la sortie du circuit combinatoire. Ce sont d'ailleurs ces délais qui permettent d'appliquer la rétroaction.

$$X(t+\Delta) = S(t) + \overline{R(t)} \cdot X(t)$$

On vérifie que le circuit est dans un état stable lorsque

$$X(t+\Delta) = X(t)$$

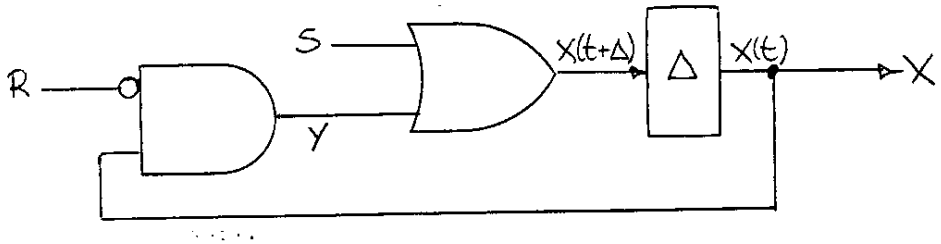


Figure 9.2 Circuit asynchrone simple : la cellule binaire

Analyse

Considérons le circuit asynchrone de la figure 9.3. On voudrait connaître la séquence de cette machine.

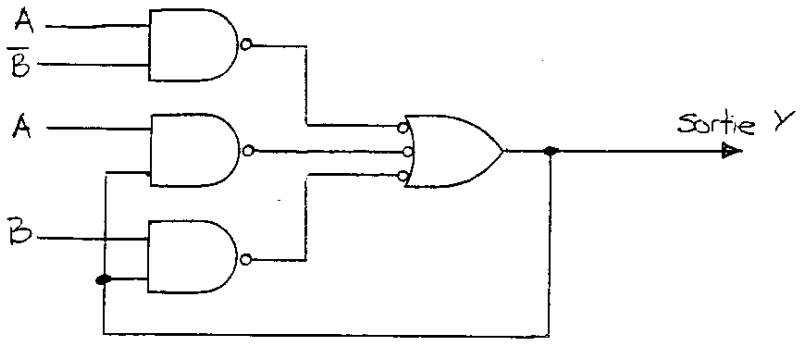


Figure 9.3 Le circuit séquentiel asynchrone à analyser

La première étape du processus d'analyse est de remodeliser le circuit de manière à bien distinguer les blocs majeurs qui le compose. Ici, séparer les 2 circuits combinatoires servant de décodage de l'état suivant et de sortie. Une fois obtenu le modèle, pour chaque état possible, deux conditions existent :

- Si $Y(t) = y(t+\Delta)$, alors le circuit est stable ou n'est pas en transition.
- Si $Y(t) \neq y(t+\Delta)$, alors le circuit est instable ou en transition d'un état vers un autre.

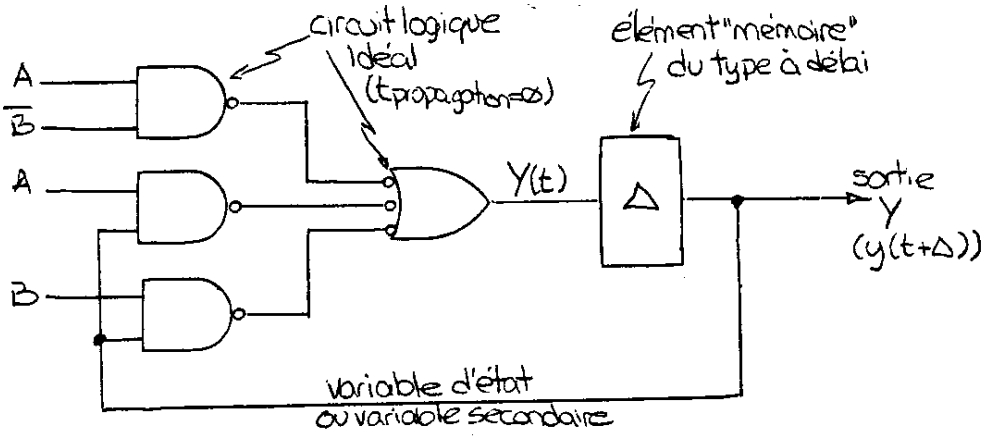


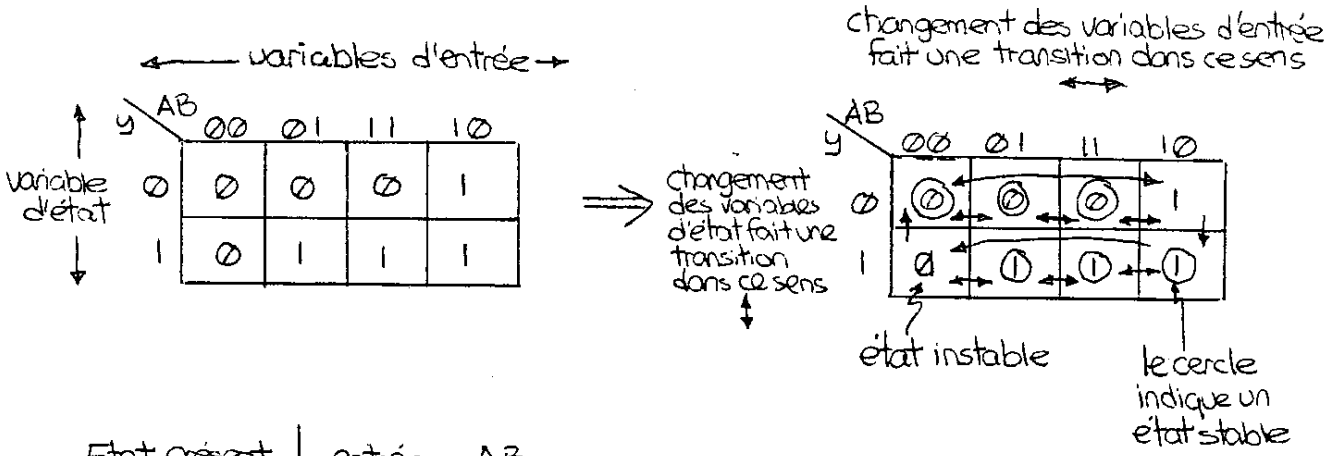
Figure 9.4 Le circuit asynchrone transformé dans son modèle fondamental

Généralement $Y(t) = f[\text{entrées, variables secondaires}]$.

Dans notre cas $Y(t) = A\bar{B} + Ay(t+\Delta) + By(t+\Delta)$, plus simplement $Y = A\bar{B} + Ay + By$.

Basé sur la nature combinatoire des changements séquentiels du circuit asynchrone, on trace une table spéciale de Karnaugh pour Y appelée la table d'excitation. Cette table donne des points de comparaison d'une façon graphique entre Y et y dans le but de déterminer la stabilité du circuit sous telles conditions d'entrée et de rétroaction; la figure 9.5 montre différents modèles de table d'excitation pour le circuit à analyser. On y voit, sur la troisième, les

états stables et ceux instables ainsi que les transitions possibles.



Etat présent y	entrées AB 00	01	11	10	
0	0	0	0	1	état suivant (Y)
1	0	1	1	1	

$$Y = A\overline{B} + Ay + By$$

Figure 9.5 Quelques modèles de table d'excitation du circuit

Une autre table d'excitation est créée, produit d'une légère modification : chaque cellule de la table de Karnaugh représente un état; à chaque état stable est assigné un nom, généralement une lettre, qui apparaît maintenant seule et encerclée; quant aux cellules des états instables, ils contiennent le nom de l'état stable vers lequel se font les transitions. C'est à partir de cette dernière table qu'on dessine aisément le diagramme d'état primitif.

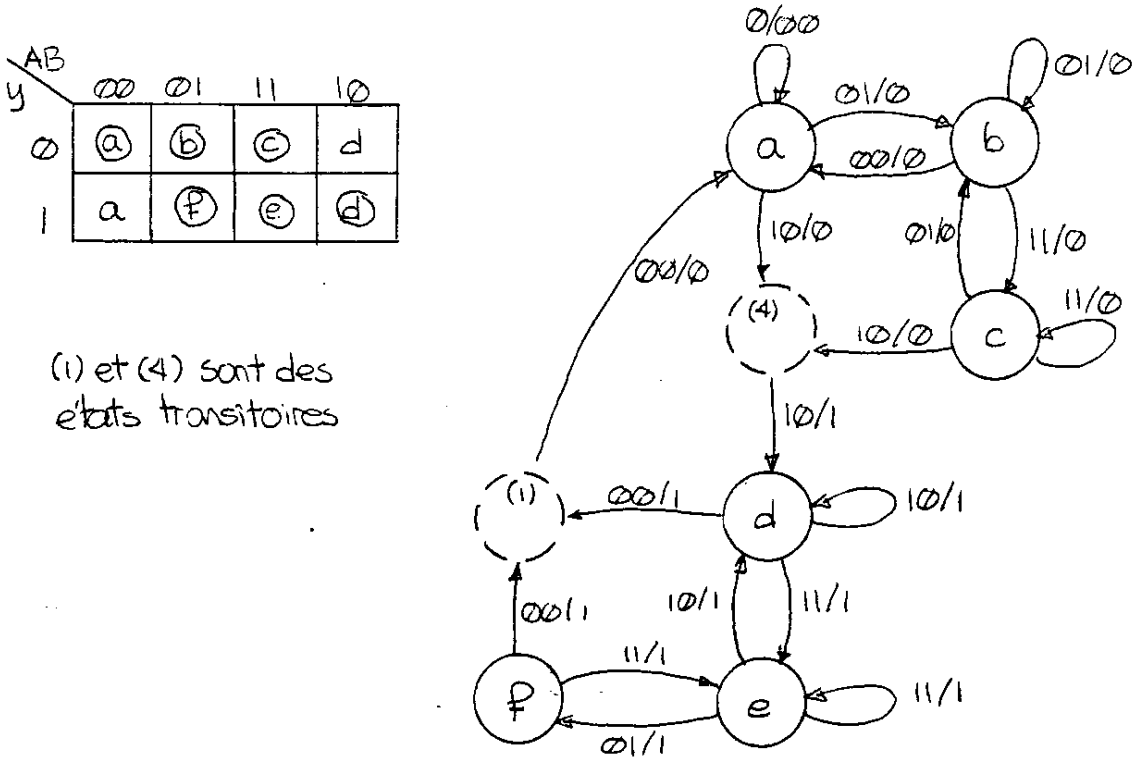


Figure 9.6 Table d'excitation encodée et diagramme d'état primitif

Evidemment, avec l'habitude, bien des étapes peuvent être abrogées mais cette technique plus rigoureuse à un but pédagogique plus subtil : celui de mieux saisir le pourquoi des démarches à établir en vue de la synthèse des circuits séquentiels asynchrones.

Exemples d'analyse

Prenons comme exemple le circuit asynchrone de la figure 9.7.

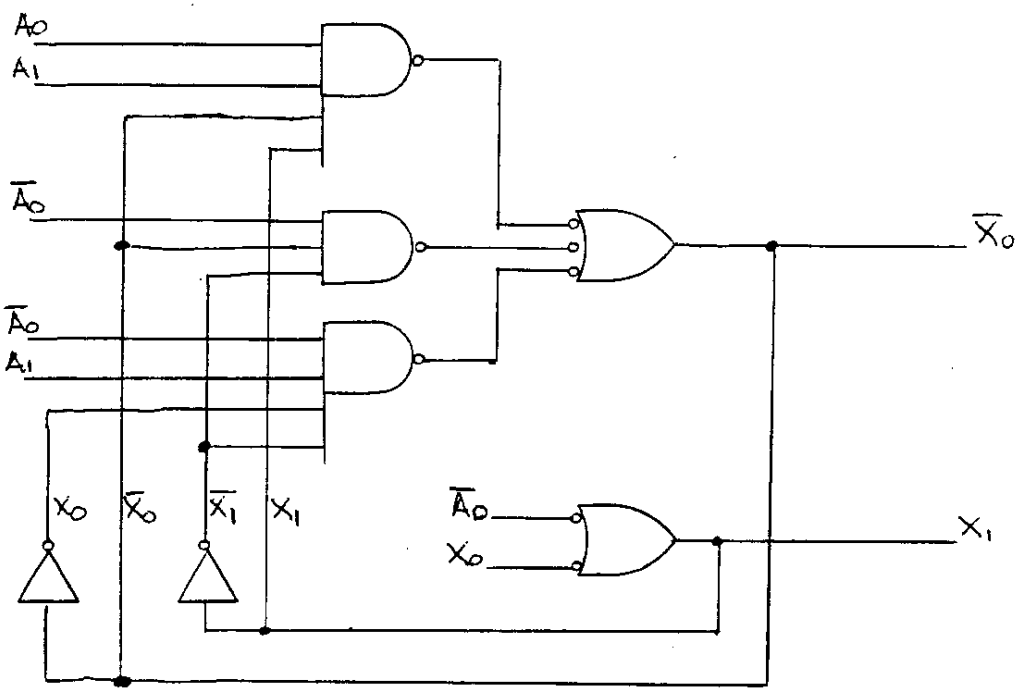


Figure 9.7 Circuit séquentiel asynchrone #2

On obtient les équations des variables d'état suivantes :

$$X_1 = \overline{X_0} + A_0$$

$$\overline{X_0} = \overline{X_1}\ \overline{X_0}\ \overline{A_0} + X_1\ \overline{X_0}\ A_1\ A_0 + \overline{X_1}\ X_0\ A_1\ \overline{A_0}$$

état suivant = f [état présent + entrées]

Après avoir dressé la table d'excitation qui donne l'état suivant en fonction de l'état présent et des entrées, on regarde les transitions possibles en supposant que l'état présent est $X_1 = 0$ et $X_0 = 0$. On remarque alors des comportements douteux.

a) Cas où $A_1 = A_0 = 0$; 1 état stable rejoint par une transition normale.

b) Cas où $A_1 = 0$ $A_0 = 1$; aucun état stable ne peut être accédé. Les sorties oscilleront dans les 4 états instables qui se bouclent.

c) Cas où $A_1 = 1$ $A_0 = 0$. Deux départs possibles à partir de l'état instable $X_1 = X_0 = 0$ mais après les transitions, 1 seul et même état stable est rejoint. Il s'agit là d'une course non critique.

d) Cas où $A_1 = A_0 = 1$. Deux départs possibles à partir de l'état instable $X_1 = X_0 = 0$ mais, plus grave encore, deux états stables peuvent être atteints selon le départ choisi. C'est une course critique.

Note : une transition directe de 0 0 à 1 1 est impossible car il ne faut changer que d'un bit à la fois.

A part ces problèmes majeurs inhérents au principe de fonctionnement d'un circuit asynchrone, d'autres problèmes se greffent tels que des transitions erronées, non prévues, causées par les aléas du circuit combinatoire agissant comme décodeur de l'état suivant.

état présent X_0X_1	entrées A_0A_1 00	01	11	10
00	01	01	11	11
01	11	11	(01)	11
11	10	10	(11)	(11)
10	(10)	00	11	11

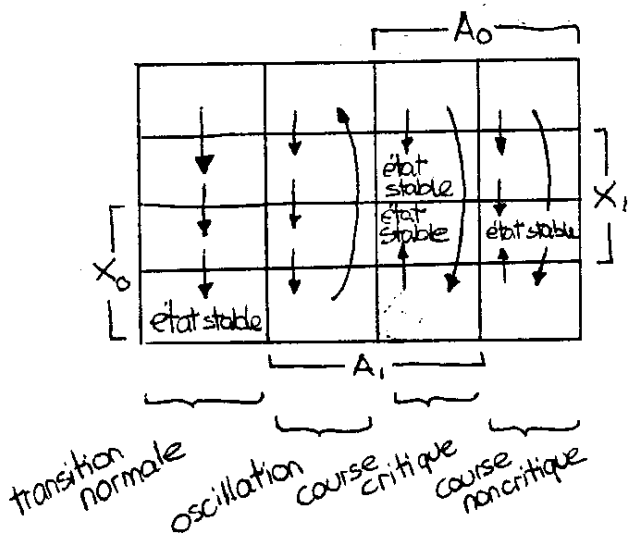


Figure 9.8 Table d'excitation et illustration des transitions dangereuses du circuit asynchrone #2

Synthèse

La synthèse est la situation inverse de l'analyse : réaliser un circuit minimum produisant une relation séquence des entrées-séquences des sorties telles que spécifiées. Une méthode fondamentale de design, celle de Hoffman, est utilisée pour la synthèse des bascules et autres circuits séquentiels de base. Elle utilise la structure proposée par la figure 9.1.

Les limitations de départ sont les suivantes :

- les décodeurs d'état suivants et de sortie sont des circuits combinatoires à 2 couches;
- le décodeur d'état suivant doit êtres sans aléa;
- les entrées et les variables d'état sont telles qu'une seule change qu'à chaque transition. Donc, tout état suivant est adjacent logique à l'état présent. Cette règle est connue sous le nom de distance unitaire de transition.

Les étapes de conception qui suivent, présentent un processus logique de développement d'un circuit séquentiel asynchrone. Un exemple servira à détailler chaque procédure.

a) Recevoir les spécifications.
b) Développer un diagramme d'état primitif ou de fonctionnement.
c) Créer une table d'état primitive appelée aussi table initiale des phases.
d) Fusionner les états redondants de la table d'état primitive pour former la table d'état réduite.
e) Faire le codage des états.
f) Transformer la table d'état réduite en une table d'excitation.
g) Dériver les expressions pour obtenir les variables d'états en lisant la table d'excitation puis faire la synthèse du décodeur d'état suivant.
h) Faire la table de Karnaugh pour le décodeur de sortie et faire la synthèse.

Exemple de synthèse

a) Spécifications :

On veut détecter, à partir de 2 cellules photo-détecteurs, la direction du mouvement d'objets assez larges (plus larges que la distance séparant les 2 cellules). Le circuit doit posséder 2 sorties indiquant un mouvement à gauche ou à droite.

b) Diagramme de fonctionnement :

La description reçue du système est souvent incomplète ou vague du point de vue concepteur. Il faut traduire la définition par un diagramme d'état primitif (mieux) ou de fonctionnement où seules les séquences possibles des entrées sont indiquées. Toutes les entrées doivent être analysées à l'intérieur d'une séquence de sorte qu'il soit possible d'obtenir 2 états qui ont les mêmes spécifications à l'entrée et à la sortie mais qui ne peuvent être classés équivalents car ils font partie de séquences indépendantes.

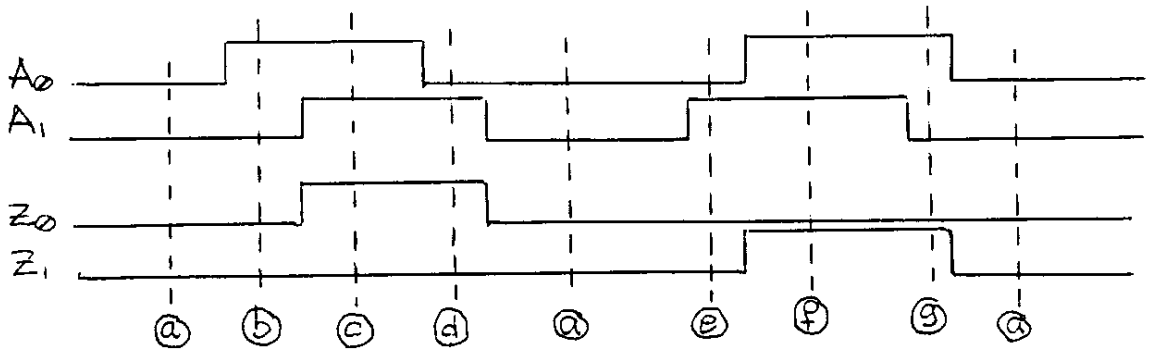


Figure 9.9 Diagramme de fonctionnement par la synthèse du circuit

c) Table d'état primitive

Dans le diagramme de fonctionnement, on a défini les états internes nécessaires pour distinguer les séquences des entrées. Dans ce premier choix, il peut y avoir des redondances mais elles ne sont pas importantes pour le moment. En réalité, les difficultés sont de reconnaître une séquence déjà définie et d'arrêter la description du problème.

Le code inscrit dans chacune des cases correspond au code du prochain état rejoint après le temps de propagation approprié. Il est important de se rappeler que si l'entrée Y (cf. problème à analyser) fournit une sortie identique y, alors les conditions d'entrée et de rétroaction sont dans un état stable. Mais les mêmes entrées ne produisent pas nécessairement les mêmes sorties, compte tenu du moment : c'est un circuit séquentiel.

La table d'état primitif indique, selon les commandes d'entrée, l'état suivant et les sorties. Il est à noter que dans cette table, il doit y avoir un seul état stable atteint à partir de l'état présent, donc un seul état stable par ligne.

état présent	A_0A_1 00	01	11	10	sorties Z_0	Z_1
(a)	(a)	e	—	b	0	0
(b)	×	—	c	(b)	0	0
(c)	—	d	(c)	×	1	0
(d)	a	(d)	×	—	1	0
(e)	×	(e)	f	—	0	0
(f)	—	×	(f)	g	0	1
(g)	a	—	×	(g)	0	1

— indique un changement impossible de 2 entrées
× indique un changement impossible d'après le problème (objet large)

Figure 9.10 Table d'état primitive pour la synthèse du circuit

d) Réduction des états

Loi de réduction : deux états peuvent être réduits à un si tous les états suivants définis dans la table d'état primitive sont égaux. (S'il n'y a pas de conflit de numéro d'état dans toutes les colonnes : chaque colonne contient les 2 mêmes états ou un état avec un état indifférent.)

Dans notre exemple de synthèse, les rangées correspondant aux états e et f ont des états suivants identiques tels que spécifiés par la règle de réduction. On peut donc combiner ces deux lignes. A ce moment, il peut y avoir plusieurs sorties possibles pour un seul état réduit, mais le processus de réduction assure des transitions normales ou course non critique.

état présent	A_0A_1 00	01	11	10
(e)	—	(e)	f	—
(f)	—	—	(f)	g
devient				
(e')	—	(e)	(f)	g

Figure 9.11 Réduction des 2 états e et f

L'examen de la table d'état primitive montre qu'une réduction à 3 états devient possible.

a ⟶ a'
b,c,d ⟶ b'
e,f,g ⟶ c'

état présent	A_0A_1 00	01	11	10
(a')	(a)	e	—	b
(b')	a	(d)	(c)	(b)
(c')	a	(e)	(f)	(g)

Figure 9.12 Nouvelle table d'état ou table d'état réduite

Ces 3 états stables seront donc produits par au moins 2 variables d'état avec rétroaction.

e) Codage des états réduits

Les états réduits doivent être représentés ou codés par au moins ABS($\log_2$(n-1)+1) variables d'état avec n égal au nombre d'état du circuit séquentiel. Ce code est limité contrairement à celui appliqué sur une machine synchrone. Il est nécessaire de passer d'un état au suivant en ne changeant qu'une seule bit du code à la fois, en s'assurant qu'il n'y aura aucun aléa possible produit par le décodeur d'état suivant : une seule entrée change à la fois (distance de transition unitaire). C'est à partir du diagramme d'état réduit que se fait le codage. Dans notre exemple, il y a transition entre a' et b' et entre a' et c' seulement.

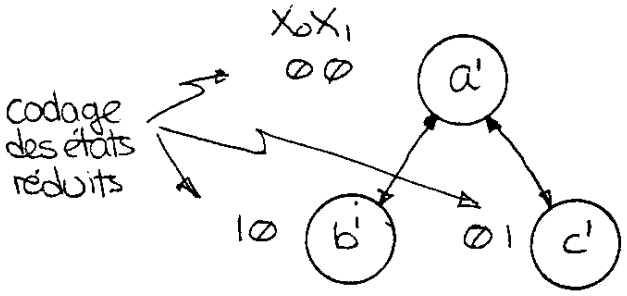


Figure 9.13 Codage des états à partir du diagramme d'état réduit

f et g) Création de la table d'excitation

Le choix des codes des variables d'état permet la création d'une table d'excitation basée sur la table d'état réduite. Cette table d'excitation, réalisée de plusieurs manières, facilite l'application des tables de Karnaugh dans le but de faire la synthèse du décodeur d'état suivant. Pour éviter les aléas, critiques au niveau du décodeur d'état suivant car ils risqueraient de modifier la séquence, il faut utiliser les règles des groupements redondants de liaison lors de la résolution des tables de Karnaugh dressées pour chacune des variables d'état.

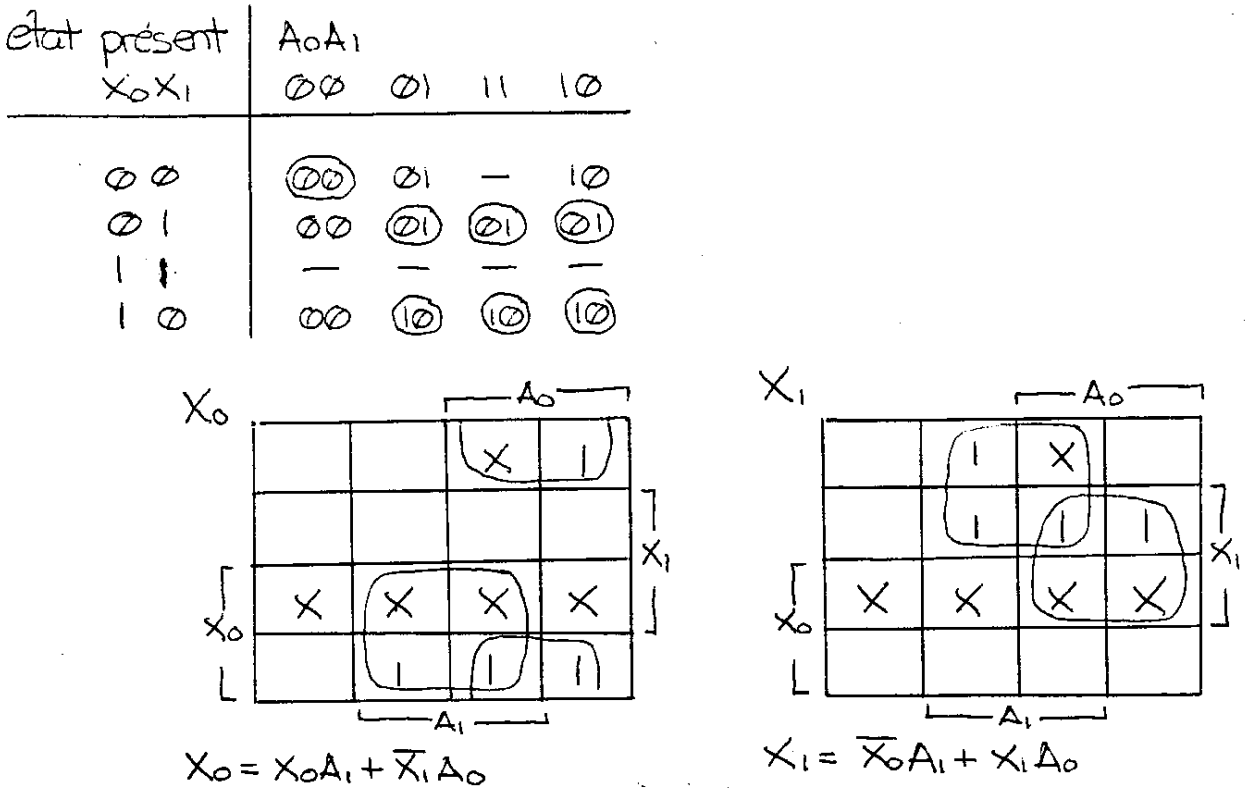


Figure 9.14 Table d'excitation et synthèse du décodeur d'état suivant sans aléa

$$X_0 = X_0\ A_1 + \overline{X}_1\ A_0$$

$$X_1 = \overline{X}_0\ A_1 + X_1\ A_0$$

h) Synthèse du décodeur de sortie

Dans un circuit séquentiel synchrone ou asynchrone, les sorties dépendent des états initiaux et des commandes d'entrées. Les tables de Karnaugh, une pour chaque variable de sortie, sont reliées aux états initiaux. (Les états initiaux sont les états avant la réduction.) Il est donc utile de se bâtir un outil montrant, pour chaque code et commandes possibles, quels sont les états initiaux afin de déduire rapidement, dans une deuxième phase, les tables de Karnaugh des variables de sortie. Cet outil s'appelle un dictionnaire.

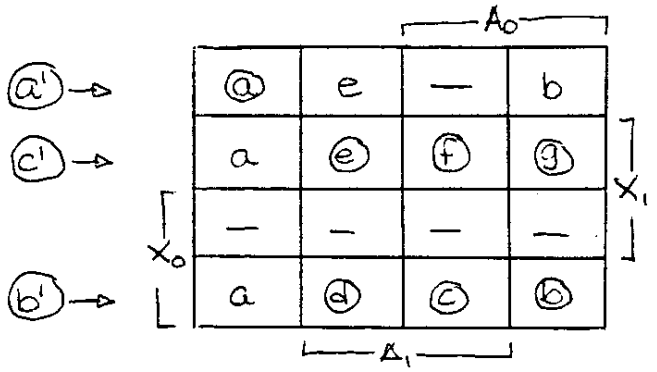


Figure 9.15 Dictionnaire de sortie pour la synthèse du circuit

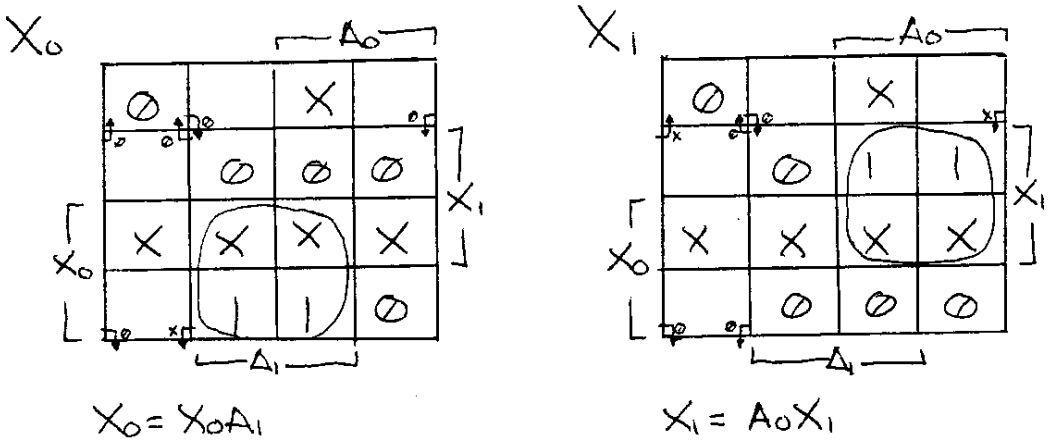
Les sorties pour les états impossibles sont indifférentes. Dans ce cas-ci, l'état impossible est $X_1\ X_0$ = 11. Les sorties pour les états stables sont spécifiées dans la table d'état primitive. Les sorties pour les états transitoires sont indifférentes lorsqu'on n'a pas à s'inquiéter des aléas (il se peut qu'un décodeur de sortie avec possibilité d'aléas soit permis). Sinon, il faut considérer toutes les transitions possibles et analyser en vue d'éliminer les aléas possibles lors des transitions non stables.

On obtient :

$$Z_0 = X_0 A_1$$

$$Z_1 = X_1\ A_0$$

et il n'y a pas d'aléa possible. Note : la seule façon de vérifier un circuit avec rétroaction consiste à ouvrir les boucles.



Note: les cases laissées blanches sont indifférentes lorsqu'on ne s'occupe pas des aléas dans le décodeur de sortie.

Figure 9.16 Tables de Karnaugh des variables de sortie en vue de réaliser le décodeur de sortie

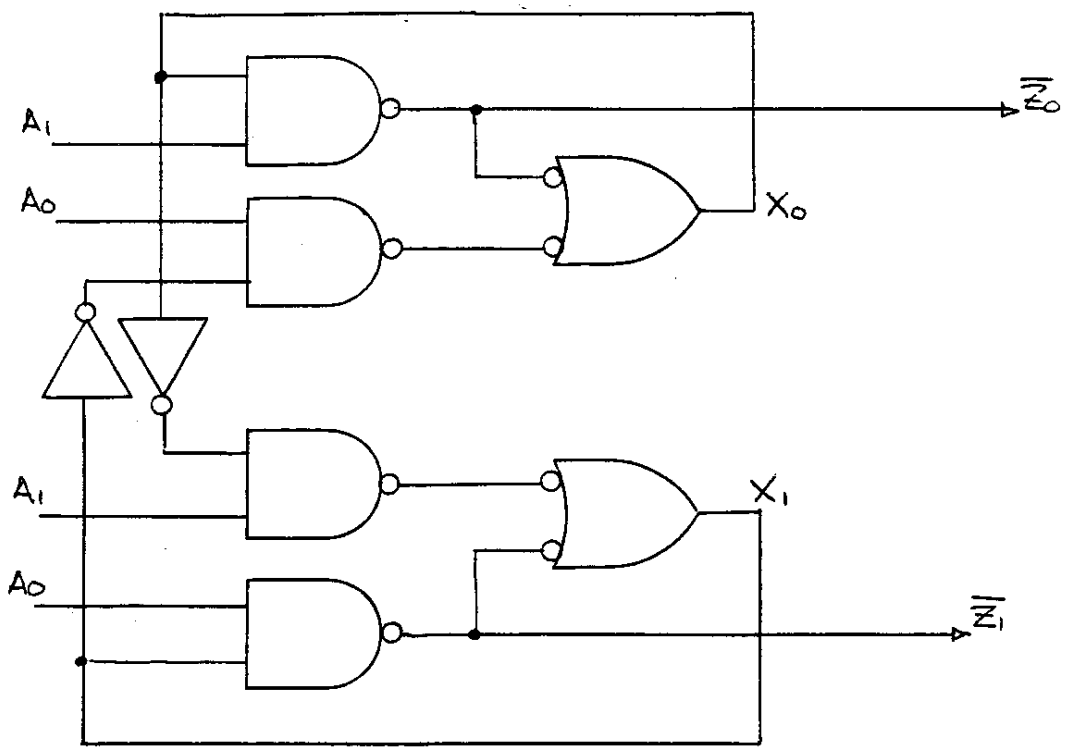


Figure 9.17 Circuit asynchrone détectant la direction d'un mouvement

Exercice

Réalisez une bascule T (toggle) synchrone. Cette bascule possède une seule entrée, l'horloge H, une seule sortie Q qui bascule (sous-entendre toggle) à chaque coup d'horloge. Vous trouverez le résultat à la page

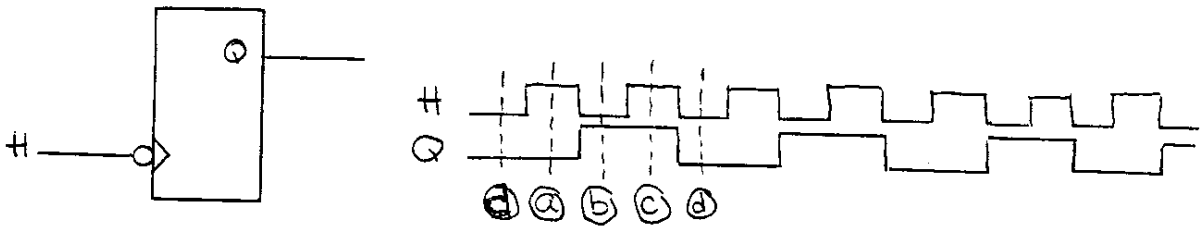


Figure 9.18 Diagramme de fonctionnemenent de la bascule T

Exemples de synthèse

"Gated oscillator"

Une onde carrée A_0 est commandée par un signal asynchrone A_1. La sortie Z est égale à A_0 pour un nombre entier d'impulsions. Si A_1 devient 1 pendant que A_0 est à 1, la sortie reste nulle jusqu'à l'impulsion suivante. Si A_1 devient 0 pendant une impulsion, alors la sortie termine une impulsion complète.

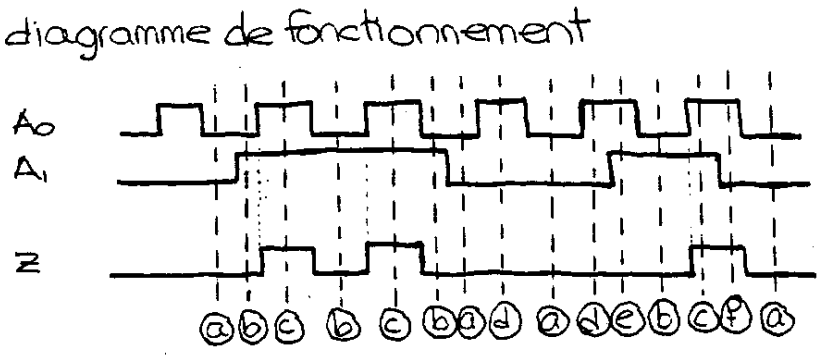


Figure 9.19 Synthèse d'un "gated oscillator"

table d'état primitive

état présent	A_0A_1 00	01	11	10	z
ⓐ	ⓐ	b	—	d	0
ⓑ	a	ⓑ	c	—	0
ⓒ	—	b	ⓔ	f	1
ⓓ	a	—	e	ⓓ	0
ⓔ	—	b	ⓔ	d	0
ⓕ	a	—	c	ⓕ	1

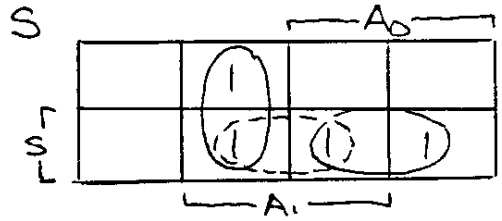
table d'état réduite

état présent	A_0A_1 00	01	11	10
ⓐ'	ⓐ	b	ⓔ	ⓓ
ⓑ'	a	ⓑ	ⓒ	ⓕ

codage trivial (1 bit → 2 états réduits)
table d'excitation

S	A_0A_1 00	01	11	10
0	⓪	1	⓪	⓪
1	0	①	①	①

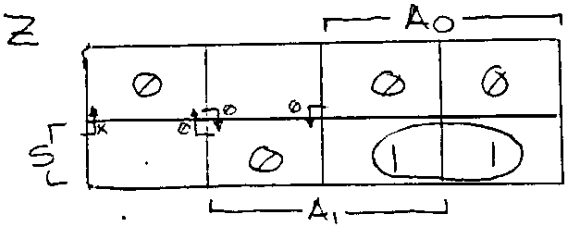
synthèse du décodeur d'état suivant
table de Karnaugh de S



$S = \bar{A}_0 A_1 + A_0 S + \underline{A_1 S}$
pour éviter les aléas

Figure 9.19 Synthèse d'un "gated oscillator"

Table de Karnaugh de la variable de sortie Z



$Z = A_0 S$

Circuit d'un "gated oscillator"

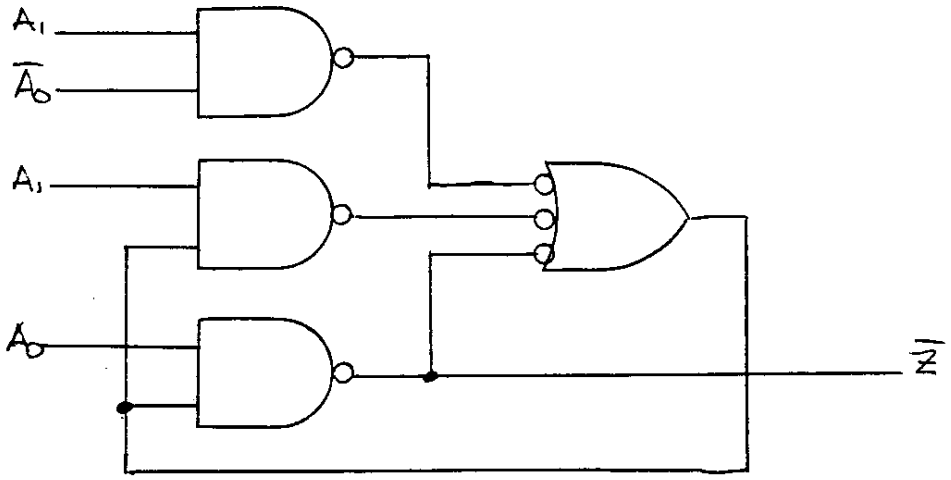


FIgure 9.19 Synthèse d'un "gated oscillator"

"One shot"

Un circuit a 2 entrées : une onde carrée A_0 et une commande asynchrone A_1. Lorsque A_1 est égale à 1, la sortie Z est une impulsion complète de A_0. On simplifie le problème en disant qu'une commande A_1 trop courte ne fait pas passer une impulsion : une impulsion passera si A_1 est égale à 1 pendant une montée de A_0.

diagramme de fonctionnement

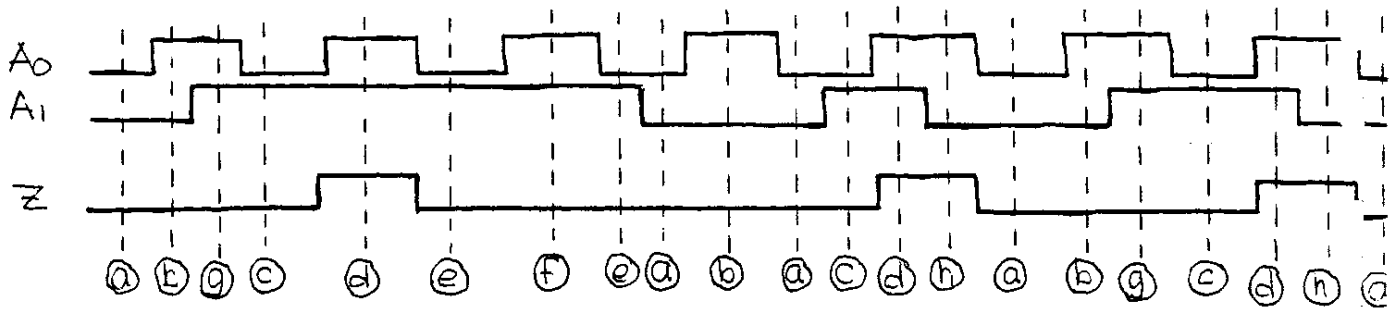


table d'état primitive

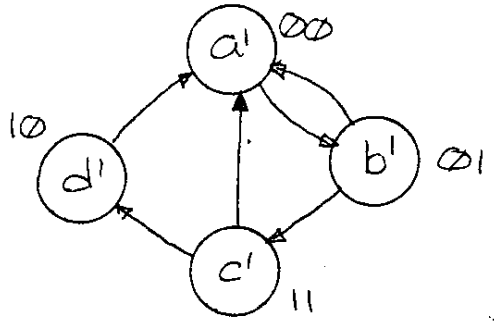
état présent	A_0A_1 00	01	11	10	Z
ⓐ	ⓐ	c	—	b	0
ⓑ	a	—	g	ⓑ	0
ⓒ	a	ⓒ	d	—	0
ⓓ	—	e	ⓓ	h	1
ⓔ	a	ⓔ	f	—	0
ⓕ	—	e	ⓕ	b	0
ⓖ	—	c	ⓖ	b	0
ⓗ	a	—	d	ⓗ	1

Figure 9.20 Synthèse d'un "one shot"

table d'état réduite

état présent	A_0A_1 00	01	11	10
a'	a	c	g	b
b'	a	e	d	—
c'	a	e	d	h
d'	a	e	f	b

codage



la transition de c' à a' est une course non critique tolérée (en changeant bit par bit on revient tout de même à a') Il est permit ici de manquer une impulsion si A_1 est trop court.

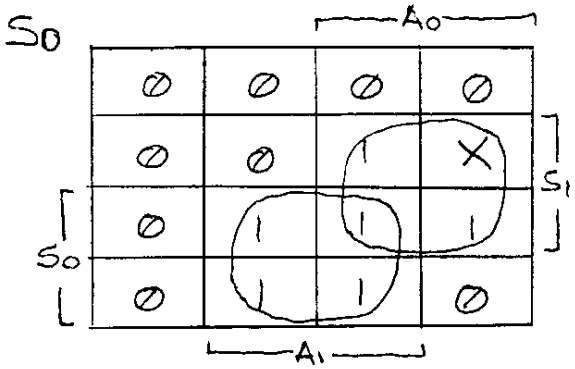
Table d'excitation

	S_0S_1	A_0A_1 00	01	11	10
a'	00	00	01	00	00
b'	01	00	01	11	—
c'	11	00	10	11	11
d'	10	00	10	10	00

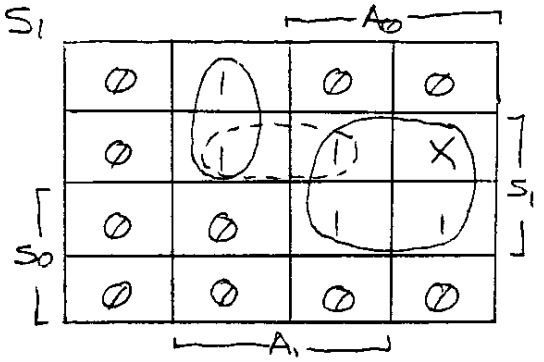
↑ démonstration de la course non critique

Figure 9.20 Synthèse d'un "one shot"

Synthèse du décodeur d'état suivant



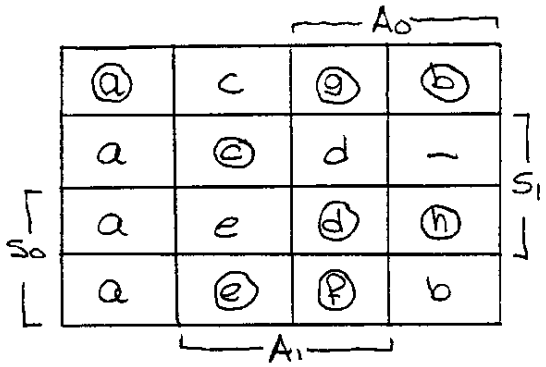
$S_0 = A_0 S_1 + A_1 S_0$

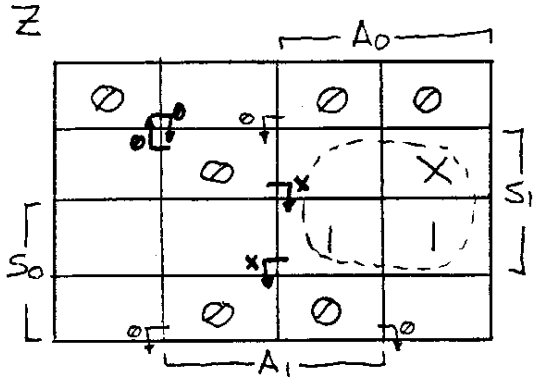


$S_1 = \bar{A}_0 A_1 \bar{S}_0 + A_0 S_1 + A_1 \bar{S}_0 S_1$

pour éviter les aléas

Table de Karnaugh des variables de sortie (z)
dictionnaire





$Z = A_0 S_1$

Figure 9.20 Synthèse d'un "one shot"

Etat transitoire supplémentaire

Il arrive parfois qu'il soit impossible de coder les états en s'assurant toujours d'une transition à distance unitaire, ceci étant vu d'après le diagramme d'état simplifié. Dans ce cas bien précis, on contourne le problème en rajoutant volontairement un état transitoire dit état transitoire supplémentaire. Cet état devra être codé comme un des états réduits, bien que transitoire même si, dans bien des cas, on doit ajouter une bit supplémentaire au codage. Le danger d'une transition par le changement de 2 bits ou plus est de provoquer des courses critiques en plus de causer des aléas.

Faisons la synthèse d'un circuit asynchrone à 2 entrées, A_0 et A_1, avec une sortie Z. Si $A_0 = 0$ et $Z = 0$, alors la première transition de 0 à 1 ou 1 à 0 de la commande A_1 avec $A_0 = 1$ met $Z = 1$ aussi longtemps que A_0 demeure à 1.

diagramme de fonctionnement

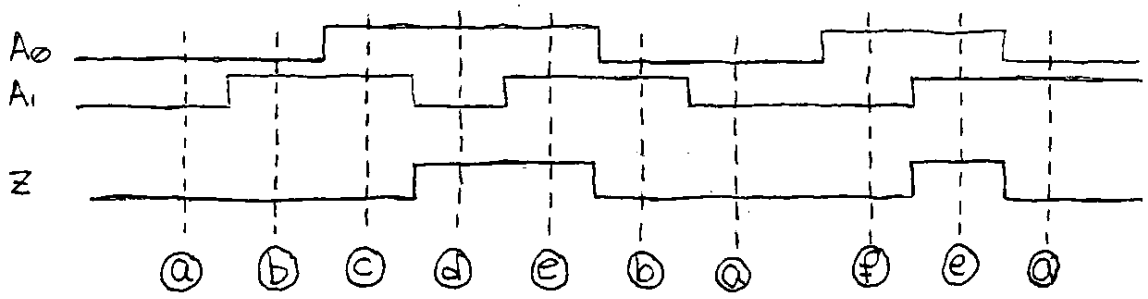


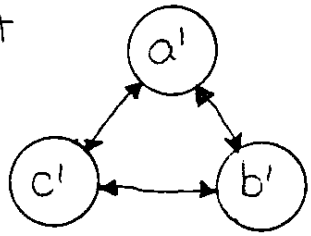
Table d'état primitive

état présent	A_0A_1 00	01	11	10	Z
(a)	(a)	b	—	f	0
(b)	a	(b)	c	—	0
(c)	—	b	(c)	d	0
(d)	a	—	e	(d)	1
(e)	—	b	(e)	d	1
(f)	a	—	e	(f)	0

Table d'état réduite

état présent	A_0A_1 00	01	11	10	
(a')	(a)	b	e	(f)	← a,f
(b')	a	(b)	(c)	d	← b,c
(c')	a	b	(e)	(d)	← d,e

diagramme d'état



un choix de code impossible

Figure 9.21 Diagramme de fonctionnement, table d'état primitive et table d'état réduite

On remarque que quelque soit le code, une transition à distance unitaire n'est pas possible pour toutes les transitions. On introduit donc l'état transitoire supplémentaire d.

Nouvelle table d'état réduite

état présent	A_0A_1 00	01	11	10
(a')	(a')	b'	d'	(a')
(b')	a'	(b')	(b')	c'
(c')	b'	b'	(c')	(c')
(d')	—	—	c'	—

Notez qu'il s'agit ici des états réduits.

d'après le diagramme d'état

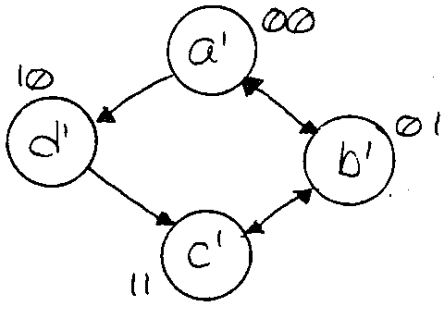
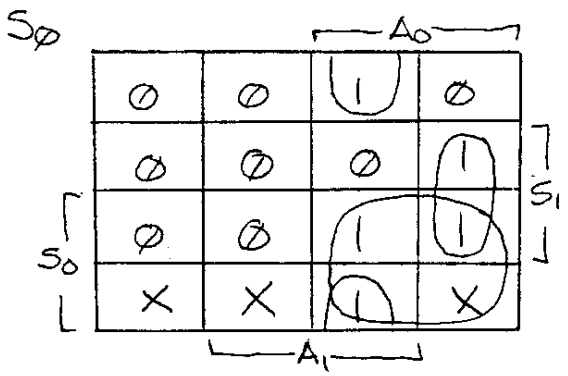


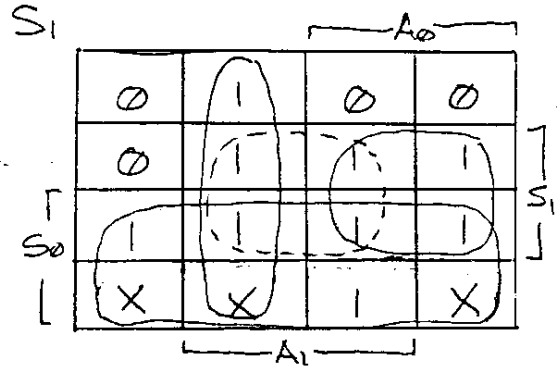
table d'excitation

état présent S_0S_1	A_0A_1 00	01	11	10
00	(00)	01	10	(00)
01	00	(01)	(01)	11
11	01	01	(11)	(11)
10	—	—	11	—

Synthèse du décodeur d'état suivant



$S_0 = A_0A_1\bar{S}_1 + A_0\bar{A}_1S_1 + A_0S_0$



$S_1 = \bar{A}_0A_1 + A_0S_1 + S_0 + A_1S_1$

pour éviter les aléas.

décodeur de sortie
on trouve $Z = S_0$

Figure 9.22 Nouvelle table d'état réduite, table d'excitation et synthèse des décodeurs avec un état transitoire supplémentaire

"cadenas à combinaison"

C'est un circuit séquentiel à 2 entrées (A_0 et A_1) et une sortie (z). La sortie est activée lorsque le circuit reçoit la séquence $A_0A_1 = \{00, 01, 11\}$ dans cet ordre seulement.

graphe

format de branchement

A_0A_1/z

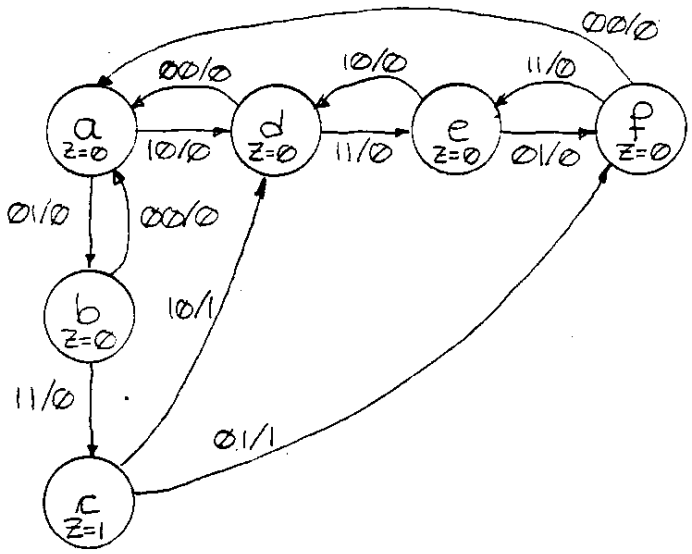


table d'état primitive

état présent	A_0A_1 00	01	11	10	z
ⓐ	ⓐ	b	—	d	0
ⓑ	a	ⓑ	c	—	0
ⓒ	—	f	ⓒ	d	1
ⓓ	a	—	e	ⓓ	0
ⓔ	—	f	ⓔ	d	0
ⓕ	a	ⓕ	e	—	0

$a' \leftarrow a, b$

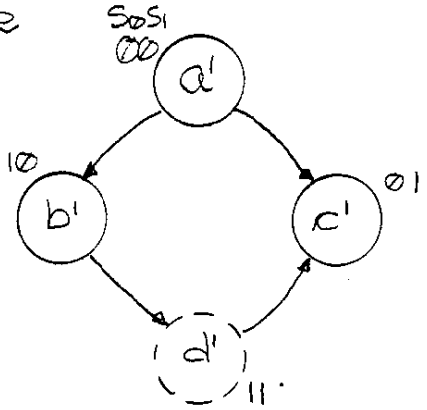
$b' \leftarrow c$

$c' \leftarrow d, e, f$

table d'état réduite

état présent	A_0A_1			
	00	01	11	10
a'	a	b	c	d
b'	–	f	e	d
c'	a	f	e	d

codage



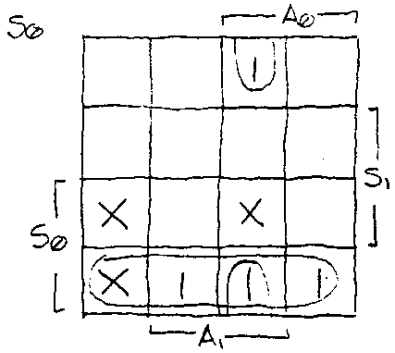
nouvelle table d'état réduite

état présent	A_0A_1			
	00	01	11	10
a'	a'	a'	b'	c
b'	–	d'	b'	d'
c'	a'	c'	c'	c'
d'	–	c'	–	c'

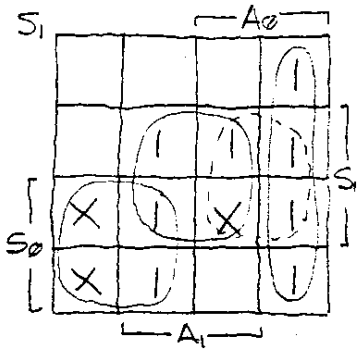
table d'excitation

état présent S_0S_1	A_0A_1			
	00	01	11	10
00	00	00	10	01
01	00	01	01	01
11	–	01	–	01
10	–	11	10	11

synthèse du décodeur d'état suivant



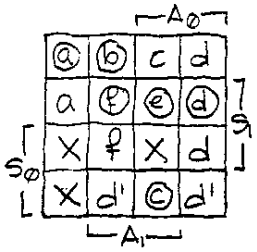
$S_0 = S_0\overline{S_1} + A_0A_1\overline{S_1}$

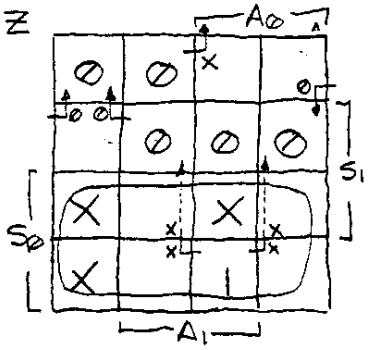


$S_1 = A_0\overline{A_1} + \overline{A_0}S_0 + A_1S_1 + \underbrace{A_0S_1}_{\text{contre les aléas}}$

décodeur de sortie

dictionnaire





$Z = S_0$

terminé le 25 mars 87
Dominic Grenier

ANNEXE "A"
FAMILLES LOGIQUES

Plusieurs circuits logiques de complexités diverses sont fabriqués de telle sorte qu'il y a compatibilité entre eux. Ils répondent alors à une standardisation commune et forment une famille.

- conception interne
- tension d'alimentation
- niveaux de fonctionnement
- sortance ("fan out")
- vitesse de propagation
- fréquence maximale d'opération
- immunité au bruit
- puissance dissipée

A la base, il existe 2 technologies → bipolaire (transistors)
→ unipolaire (transistors à effet de champ)

- bipolaire → RTL, DTL, HTL
  → TTL
  → ECL
  → I^2L
- unipolaire → MOS
  → CMOS
  → HMOS, NMOS

Les familles RTL, DTL et HTL ne servent pratiquement plus aujourd'hui mais leur compréhension sert de base pour l'analyse du TTL qui n'est qu'une modification du DTL.

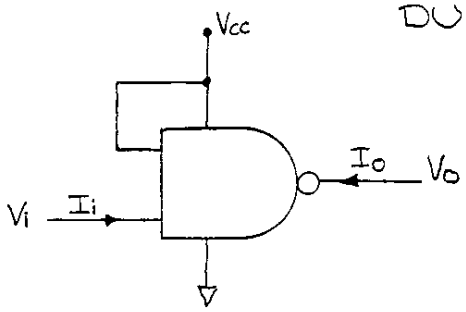
Les familles I^2L, MOS, HMOS, NMOS permettent une large intégration. Entre autre, le HMOS est la technologie des microprocesseurs : lent à ses débuts, il atteint une vitesse quasi égale à celle du TTL standard.

	RTL	DTL	HCL	TTL (Standard)	ECL	MOS	CMOS
porte de base (logique positive)	NOR	NAND	NAND	NAND	OR-NOR	NAND	NOR ou NAND
sortance	5	8	10	10	25	20	>50
puissance dissipée par porte (mW)	12	8	50	10	25-50	1	0.01 statique 1 à 1MHz
immunité au bruit		bon	excellent	très bon (0.4V)	moyen (0.1V)		très bon (~40% de V_{cc})
délai de propagation d'une porte (ns)	12	30	90	10	1-2	1000	50
nombre de fonctions	élevé	moyen		très élevé	élevé	bas	très élevé
tensions d'alimentation				V_{cc}=5V±5%	V_{EE} = -5.2V±5%		$V_{DD}-V_{SS}$ = 3-18V
niveaux d'entrée				1 : 5V à 2V 0 : 0.8V à 0	1 : -0.81 à -1.105 0 : -1.475 à -1.85		1 : $V_{DD}-V_{SS}$=5V, 3.5V(B), 4V(UB) 0 : 1.5V(B), 1V(UB) à 0
niveaux de sortie				1 : 5V à 2.4V 0 : 0.4V à 0	1 : -0.81 à -1.105 0 : -1.475 à -1.85		1 : $V_{DD}-V_{SS}$=5V à 4.95V 0 : 0.05V à 0

ANNEXE B

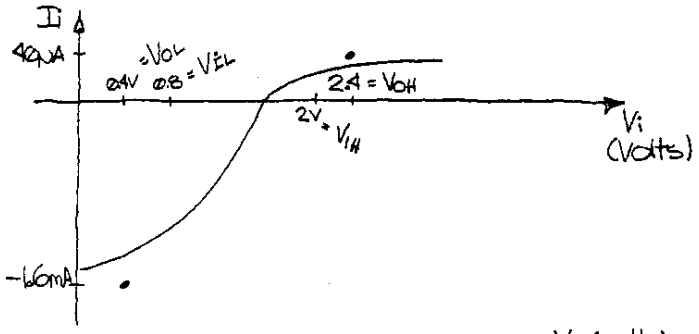
CARACTERISTIQUES ENTREE/SORTIE DU TTL STANDARD

montage :



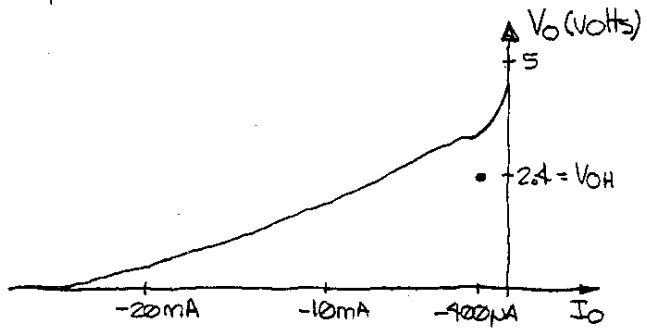
caractéristiques d'entrée :

I_{IL} max = -1.6mA (à 0.4V)
I_{IH} max = 40µA (à 2.4V)



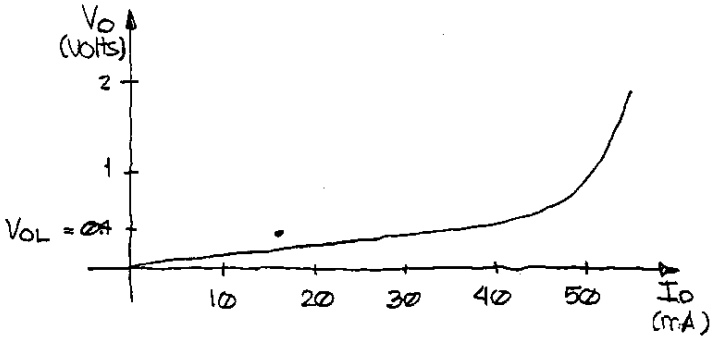
caractéristiques de sortie :
(état haut ⇒ Vi = 0)

I_{OH} max = -0.4mA (à 2.4V)
(garanti)

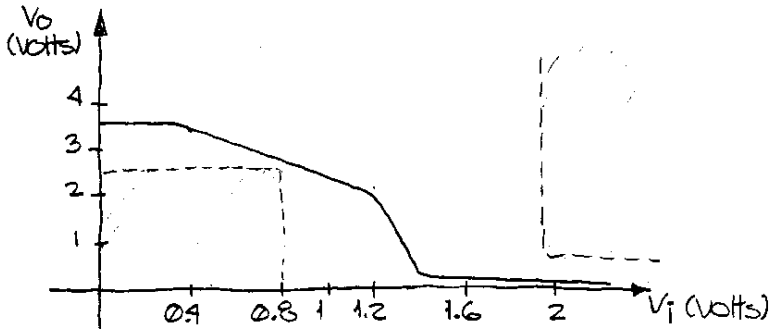


caractéristiques de sortie :
(état bas ⇒ Vi = Vcc)

I_{OL} max = 16mA (à 0.4V)
(garanti)



fonction de transfert :



ANNEXE C

OPERATIONS ARITHMETIQUES AVEC CODAGE COMPLEMENT (2 ou restreint)

Complément restreint :

soustraction et addition = un seul circuit

soustraction = addition des 2 opérandes, le soustracteur étant complémenté pour changer son signe.

attention : il faut tenir compte du "end around carry"

Exemples : codage sur 4 bits ($(-7)_{10}$ à $(7)_{10}$)

$$\begin{array}{r} +(4)_{10} \\ \underline{(3)_{10}} \\ (7)_{10} \end{array} \rightarrow \begin{array}{r} +(0100)_{2,CR} \\ \underline{(0011)_{2,CR}} \\ (0111)_{2,CR} \end{array}$$

$$\begin{array}{r} -(4)_{10} \\ \underline{(3)_{10}} \\ (7)_{10} \end{array} \rightarrow \begin{array}{r} -(0100)_{2,CR} \\ \underline{(0011)_{2,CR}} \\ \end{array} \rightarrow \begin{array}{r} +(0100)_{2,CR} \\ \underline{(1100)_{2,CR}} \\ (1)0000 \\ \hline (0001)_{2,CR} \end{array}$$

c'est le "end around carry"

$$\begin{array}{r} -(4)_{10} \\ \underline{(-3)_{10}} \\ (7)_{10} \end{array} \rightarrow \begin{array}{r} -(0100)_{2,CR} \\ \underline{(1100)_{2,CR}} \\ \end{array} \rightarrow \begin{array}{r} +(0100)_{2,CR} \\ \underline{(0011)_{2,CR}} \\ (0111)_{2,CR} \end{array}$$

$$\begin{array}{r} +(-4)_{10} \\ \underline{(-3)_{10}} \\ (-7)_{10} \end{array} \rightarrow \begin{array}{r} +(1011)_{2,CR} \\ \underline{(1100)_{2,CR}} \\ (1)0111 \\ \hline (1000)_{2,CR} \end{array}$$

dépassement : il y a dépassement si seulement si la MSB du résultat est différente de celles des opérandes lorsque ces dernières sont égales.

Exemples : $+\begin{matrix}(5)_{10}\\(3)_{10}\\(8)_{10}\end{matrix} \rightarrow +\begin{matrix}(0101)_{2,CR}\\(0011)_{2,CR}\\(1000)_{2,CR}\end{matrix}$

or la MSB des 2 opérandes = 0
mais la MSB du résultat = 1
donc dépassement

Complément 2:

soustraction et addition = un seul circuit
soustraction = addition des 2 opérandes, le soustracteur étant complémenté pour changer son signe.

Exemples: codage sur 4 bits ($(-8)_{10}$ à $(7)_{10}$)

$+\begin{matrix}(5)_{10}\\(2)_{10}\\(7)_{10}\end{matrix} \rightarrow +\begin{matrix}(0101)_{2,cv}\\(0010)_{2,cv}\\(0111)_{2,cv}\end{matrix}$

$-\begin{matrix}(5)_{10}\\(2)_{10}\\(3)_{10}\end{matrix} \rightarrow -\begin{matrix}(0101)_{2,cv}\\(0010)_{2,cv}\end{matrix} \rightarrow +\begin{matrix}(0101)_{2,cv}\\(1110)_{2,cv}\\\textcircled{1}\,0011\\(0011)_{2,cv}\end{matrix}$

pas de dépassement

$-\begin{matrix}(5)_{10}\\(-2)_{10}\\(7)_{10}\end{matrix} \rightarrow -\begin{matrix}(0101)_{2,cv}\\(1110)_{2,cv}\end{matrix} \rightarrow +\begin{matrix}(0101)_{2,cv}\\(0010)_{2,cv}\\(0111)_{2,cv}\end{matrix}$

$+\begin{matrix}(-5)_{10}\\(-2)_{10}\\(-7)_{10}\end{matrix} \rightarrow +\begin{matrix}(1011)_{2,cv}\\(1110)_{2,cv}\\1\;1001\\(1001)_{2,cv}\end{matrix}$

dépassement: il y a dépassement si seulement si la MSB du résultat est différente de celles des opérandes lorsque ces dernières sont égales.