Université de Nice-Sophia Antipolis
DEUG MIAS-MI - première année

Électronique Numérique

Cours
Second semestre

C. Belleudy, D.Gaffé

version 4.4

Chapitre 1

Les circuits arithmétiques

1.1 Additionneur

1.1.1 Principe

Considérons deux nombres $X = X_{n-1}...X_1X_0$ et $Y = Y_{n-1}...Y_1Y_0$. Pour effectuer la somme $S = X + Y$, on procède comme en décimal, en partant du rang 0 et on détermine rang par rang la somme S ; à chaque rang i on détermine :

- la somme modulo 2 de X_i, Y_i et de la retenue R_{i-1} obtenue au rang précédent, c'est le S_i de la somme S.
- la retenue R_i à reporter sur le rang i+1.

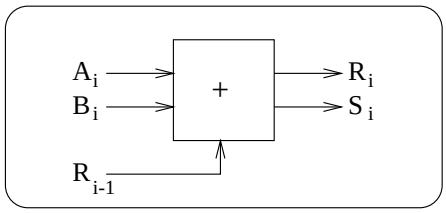
Les deux opérations sont résumées dans le tableau ci-dessous :

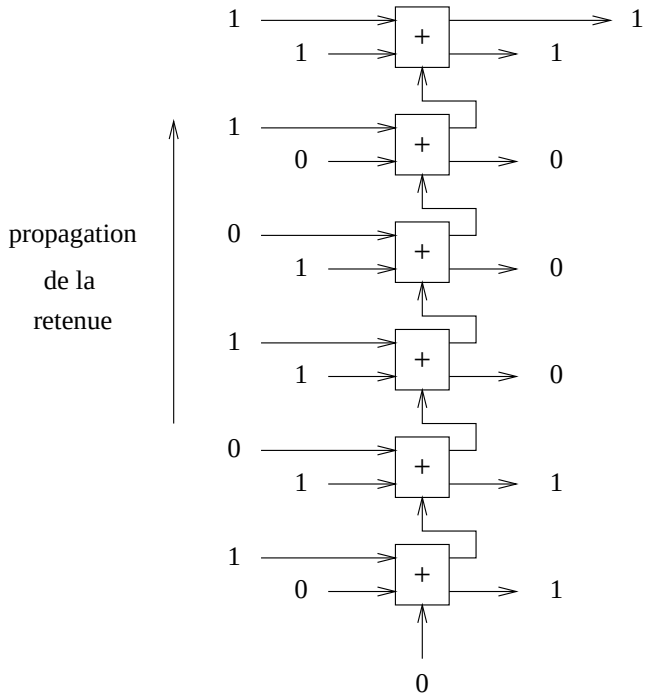
$X_iY_iR_{i-1}$	$X_i + Y_i + R_{i-1}$	R_i	S_i
000	0	0	0
001	1	0	1
010	1	0	1
011	2	1	0
100	1	0	1
101	2	1	0
110	2	1	0
111	3	1	1

Ceci donne par exemple pour six bits :

$$\begin{array}{cccccccc} & 1 & 1 & 1 & 1 & & & \\ & & 1 & 1 & 0 & 1 & 0 & 1 \\ & & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 1 & \end{array}$$

Comme les manipulations binaires sont les mêmes pour chaque bits, nous concevons l'additionneur élémentaire ci-dessous et nous le dupliquons autant de fois que nécessaire.

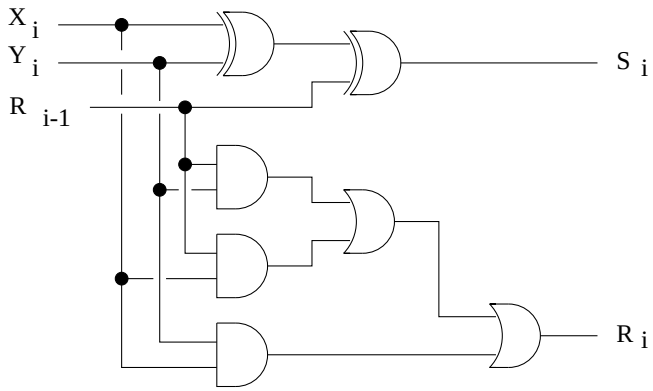




Après simplification par Karnaugh (damier sur S_i), nous obtenons les équations :

$$\begin{aligned} S_i &= X_i \oplus Y_i \oplus R_{i-1} \\ R_i &= X_i.R_{i-1} + Y_i.R_{i-1} + X_i.Y_i \end{aligned}$$

et par conséquent le schéma suivant :



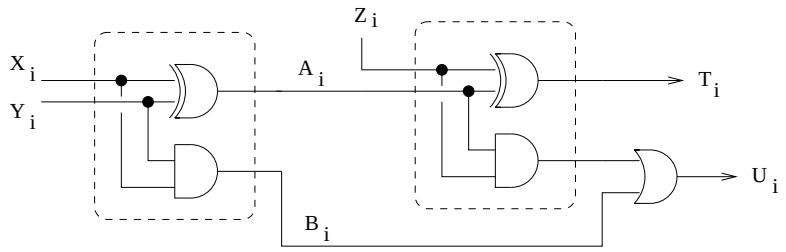
Remarque : R_i peut aussi s'écrire $R_{i-1}.(X_i + Y_i) + X_i.Y_i = R_{i-1}.P_i + G_i$

avec $P_i = X_i + Y_i$ et $G_i = X_i.Y_i$

Le terme G_i est alors appelé *terme de génération* car sa valuation à 1 est suffisante pour forcer la retenue à 1. Le terme P_i est appelé *terme de propagation* car il propage ou non la retenue précédente.

1.1.2 Demi-additionneur

Considérons le schéma constitué de deux cellules identiques :



On peut remarquer que :

$$\begin{aligned} A_i &= X_i \oplus Y_i \\ B_i &= X_i.Y_i \\ C_i &= A_i.Z_i \\ T_i &= A_i \oplus Z_i \\ U_i &= B_i + C_i \end{aligned}$$

d'ou

$$T_i = X_i \oplus Y_i \oplus Z_i$$

$$\begin{aligned} U_i &= X_i.Y_i + A_i.Z_i \\ &= X_i.Y_i + (X_i \oplus Y_i).Z_i \\ &= X_i.Y_i + (X_i.\overline{Y_i} + \overline{X_i}.Y_i).Z_i \\ &= X_i.Y_i + X_i.\overline{Y_i}.Z_i + \overline{X_i}.Y_i.Z_i \\ &= X_i.Y_i + X_i.Y_i.Z_i + X_i.Y_i.Z_i + X_i.\overline{Y_i}.Z_i + \overline{X_i}.Y_i.Z_i \\ &= X_i.Y_i + X_i.Z_i + Y_i.Z_i \end{aligned}$$

Par identification avec les équations l'additionneur élémentaire, nous obtenons :

$$\begin{aligned} Z_i &= R_{i-1} \\ T_i &= S_i \\ U_i &= R_i \end{aligned}$$

Donc ce montage est également un additionneur élémentaire : Chaque cellule est appelé demi-additionneur.

1.1.3 Les indicateurs de validité

En arithmétique binaire, il existe classiquement deux indicateurs :
- CARRY (retenue) : la dernière retenue a été levée
- OVERFLOW (dépassement de capacité) : Le nombre obtenue a une taille plus grande que le format qui lui était attribué.

Sur les nombres non-signés sur n bits, le carry et l'overflow représente la même notion car la retenue la plus à gauche sort justement du format.

d'ou $carry = overflow = R_{n-1}$ (avec les bits codés de 0 à $n-1$).

Sur les nombres signés sur n bits, $carry = R_{n_1}$ par définition.

Par contre il y a dépassement de capacité si la somme de deux nombres positifs devient négative ou la somme de deux nombre négatifs devient positive. Comme le signe est donné pas le bit le plus à gauche, le premier cas s'écrit : $\overline{X_{n-1}}.\overline{Y_{n-1}}.S_{n-1}$ et le second $X_{n-1}.Y_{n-1}.\overline{S_{n-1}}$.

d'ou $overflow = \overline{X_{n-1}}.\overline{Y_{n-1}}.S_{n-1} + X_{n-1}.Y_{n-1}.\overline{S_{n-1}}$.

1.1.4 Additionneur à retenues anticipées

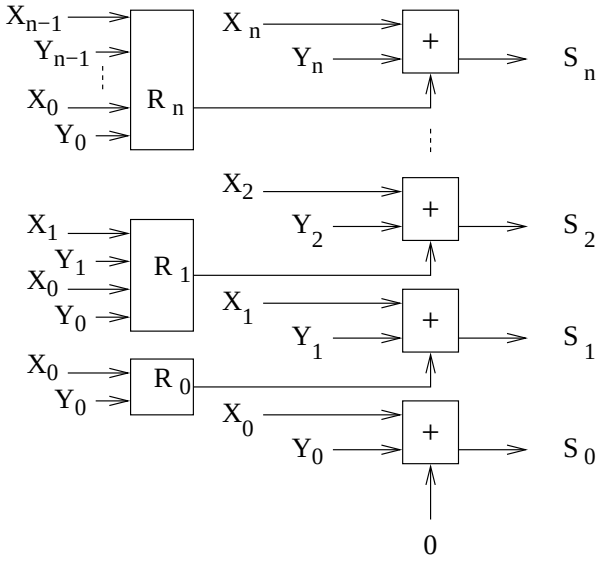
Sur un additionneur classique à n bits, le temps de calcul global est conditionné par le temps de propagation de la retenue de la première cellule élémentaire jusqu'à la dernière.

Si nous considérons un temps T de propagation dans une porte logique quelle que soit celle-ci, nous obtenons :

nombre de cellules	temps de S_n avec portes à 2 entrées	temps de R_n avec portes à 2 entrées	temps de S_n avec portes à 3 entrées (sauf OU-exclusif)	temps de R_n avec portes à 3 entrées
1	$2T$	$3T$	$2T$	$2T$
2	$5T$	$6T$	$4T$	$4T$
3	$8T$	$9T$	$6T$	$6T$
n	$3nT-1$	$3nT$	$2nT$	$2nT$

Le besoin d'additionneurs de plus en plus performants, a conduit à une nouvelle solution qui anticipe la valeur de la retenue dans les calculs au lieu d'attendre sa propagation normale.

Le schéma qui suit montre la structure d'un tel additionneur. On peut se douter qu'il nécessite beaucoup plus de portes logiques :



Pour concevoir les modules spéciaux d'anticipation de la retenue, il suffit de revenir à l'équation

d'origine de celle-ci et d'appliquer la recursion :

$$\begin{aligned} R_0 &= X_0.Y_0 \\ R_i &= X_i.R_{i-1} + Y_i.R_{i-1} + X_i.Y_i \ \ \forall i > 0 \end{aligned}$$

d'ou $R_1 = X_1.R_0 + Y_1.R_0 + X_1.Y_1 = X_1.X_0.Y_0 + Y_1.X_0.Y_0 + X_1.Y_1$

de même

$$\begin{aligned} R_2 &= X_2.R_1 + Y_2.R_1 + X_2.Y_2 \\ &= X_2.(X_1.X_0.Y_0 + Y_1.X_0.Y_0 + X_1.Y_1) + Y_2.(X_1.X_0.Y_0 + Y_1.X_0.Y_0 + X_1.Y_1) + X_2.Y_2 \\ &= X_2.X_1.X_0.Y_0 + X_2.Y_1.X_0.Y_0 + X_2.X_1.Y_1 + Y_2.X_1.X_0.Y_0 + Y_2.Y_1.X_0.Y_0 + Y_2.X_1.Y_1 + X_2.Y_2 \end{aligned}$$

et ainsi de suite...

On remarque que les équations deviennent rapidement complexes, mais elles peuvent toujours s'écrire comme une somme de produits. Donc leur temps d'évaluation est constant quel que soient leur complexité ($= 2T$). L'évaluation global de S_i prend alors un temps de :

$$\boxed{2T \text{ (retenue } R_{i-1}) + 2T \text{ (cellule } i)}$$

Conclusion :

- $T_{add} = 2nT$: additionneur classique à n cellules
- $T_{add} = 4T$: additionneur à n cellules avec retenues anticipées

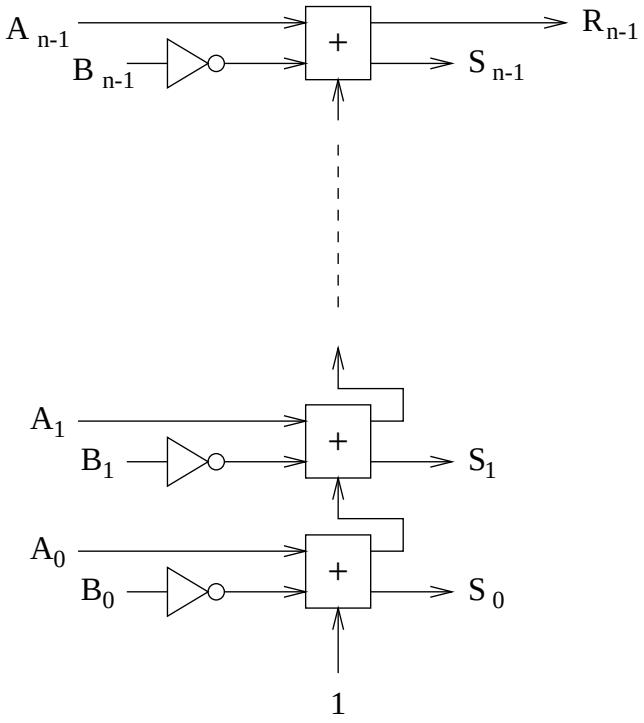
Dans le cas de l'additionneur à retenue anticipée, le temps de calcul ne dépend plus du nombre n de cellules : il est constant. Ceci est particulièrement avantageux pour les processeurs actuels qui manipulent 32 bits de données voire plus.

Le miracle vient du fait que beaucoup de calculs se font simultanément en parallèle. Le prix à payer est une explosion du nombre de portes logiques ; mais est-ce encore un problème aujourd'hui avec le taux d'intégration actuel de transistors sur un substrat de silicium ?

1.2 Soustraction

Dans un souci d'économie de circuits, la soustraction A-B est réalisée, dans la majorité des circuits numériques, comme une addition entre A et la représentation en complément à deux de B (A+(-B)). Le complément à deux de B est facile à obtenir car il suffit d'inverser B et de lui ajouter globalement "1" et nous pouvons nous appuyer sur la propagation de la retenue en choisissant $R_{-1} = 1$ au lieu de $R_{-1} = 0$!

d'ou le schéma suivant :



1.3 Multiplication non signée

La méthode de base est calquée sur celle utilisée en décimal ; suivant que le chiffre du multiplicateur vaut 1 ou 0, on ajoute ou non le multiplicande à la somme partielle, en décalant à chaque fois celui-ci d'un rang vers la gauche. En effet, chaque décalage à gauche, induit une multiplication par la base (ici 2).

A			1	0	1	0	
B		×	1	0	1	1	
			1	0	1	0	
		1	0	1	0		
	0	0	0	0			
1	0	1	0				
1	1	0	1	1	1	0	

1.4 Division non signée

Là encore, la méthode de base consiste à essayer, comme en décimal, de soustraire le diviseur, multiplié ici par leur seul facteur possible : 1.

Pour les quatre opérations décrites ci-dessus, les méthodes exposées ne constituent que des procédures de base, à partir desquelles on a imaginé un grand nombre de raffinements, destinés à en accroître la vitesse d'exécution, lorsqu'elles sont mises en oeuvre dans un calculateur ou système arithmétique.

Chapitre 2

Générateur de fonctions et de mintermes, Multiplexeurs, Démultiplexeurs

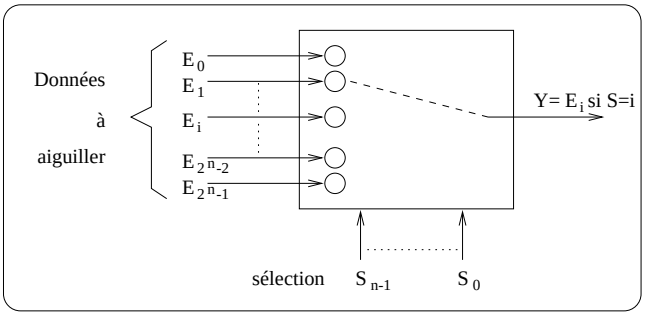
2.1 Introduction

Dans tous les systèmes numériques où les traitements sur les informations sont effectués, il est nécessaire de les aiguiller suivant la fonction à réaliser. Le principe d'aiguillage du trafic ferroviaire représente le modèle type de la notion de transfert d'informations. Plusieurs trains peuvent circuler successivement sur une même voie en provenance de lieux différents et transitant vers des destinations distinctes. Ces notions d'aiguillage sont les éléments de base du multiplexage et démultiplexage numérique.

2.2 Le multiplexeur

2.2.1 La fonction de multiplexage

Le multiplexage est une opération qui consiste à faire circuler sur un seul conducteur des informations provenant de sources multiples.



A partir de cette présentation fonctionnelle du multiplexeur, on va maintenant déterminer la fonction logique réalisée par ce type de circuit. Considérons la table de vérité d'un multiplexeur à deux entrées :

S_1S_0	Y
00	E_0
01	E_1
10	E_2
11	E_3

Cette table nous permet de déterminer l'équation logique de Y :

$$Y = m_0.E_0 + m_1.E_1 + m_2.E_2 + m_3.E_3$$

où m_i (minterme numéro i) représente la fonction qui vaut 1 pour la combinaison d'entrée $(S_1S_0)_2 = (i)_{10}$. C'est à dire :

$$Y = \overline{S_1}.\overline{S_0}.E_0 + \overline{S_1}.S_0.E_1 + S_1.\overline{S_0}.E_2 + S_1.S_0.E_3$$

En généralisant ce raisonnement, l'équation logique d'un multiplexeur à n entrées de sélection s'écrit :

$$Y = \sum_{i=0}^{2^n-1} E_i m_i$$

2.2.2 Générateur de fonction

Toute fonction logique combinatoire peut se mettre sous la forme canonique disjonctive :

$$F(X_1, X_2, ..., Xn) = \sum_{i=0}^{2^n-1} F(i) m_i$$

où $X_1, ..., X_n$ représentent les variables d'entrées, $F(i)$ la valeur de la fonction lorsqu'on considère $X_1...X_n$ comme le codage binaire de i et m_i la fonction binaire qui vaut 1 seulement pour la combinaison $X_1...X_n$ considérée (minterme associé à la combinaison $X_1...X_n$). En comparant l'équation précédente et celle établie pour la sortie d'un multiplexeur à n entrées, il est facile de voir qu'il est possible de réaliser toutes les fonctions de n variables : les entrées de sélection du multiplexeur sont alors les variables de la fonction et les entrées de données du multiplexeur permettent de sélectionner la fonction à réaliser. Prenons à titre d'exemple la table de vérité présentée ci-dessous qui possède trois variables d'entrées A, B, C et considérons un mpx $8 \rightarrow 1$ sur lequel les entrées du système traité sont connectées aux entrées de sélection du mpx.

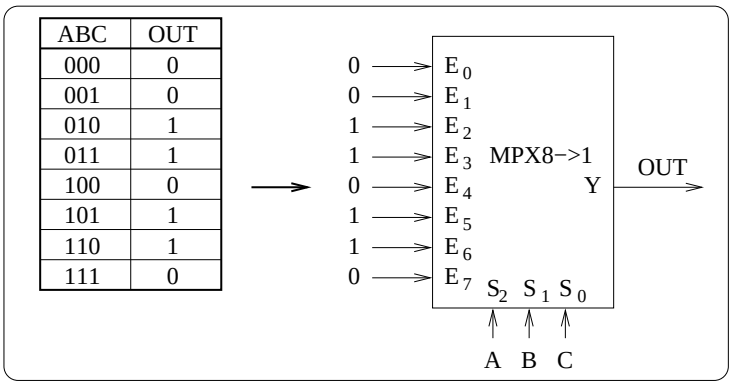


FIG. 2.1 – Exemple de cablâge de multiplexeur obtenu directement à partir de la table de vérité

Lorsque le nombre d'entrées du système étudié devient grand, il n'est pas toujours aisé de disposer d'un multiplexeur avec autant d'entrées de sélection que de variables d'entrées. Dans ce cas, le concepteur peut essayer de trouver une solution moins onéreuse en utilisant un multiplexeur avec un plus petit nombre d'entrée de sélection et des portes logiques. Pour ce faire, il faut dans un premier temps choisir le sous-ensemble des variables d'entrées qui seront directement assimilées aux entrées

de sélection ; ces variables sont alors appelées variables de sélection.

Le cablage des entrées de multiplexage ($E_0, E_1, ...E_n$) est obtenue par synthèse des tables de vérité qui peuvent être établies pour chaque combinaison des variables de sélection.

Considérons ainsi l'entrée E_i : la table de vérité du système étudié peut être réduite aux seules combinaisons où les variables représentant les entrées de sélection valent i. Les équations logiques obtenues ne dépendent alors que des variables qui ne sont pas connectées aux entrées de sélection du multiplexeur.

Pour illustrer ce mode de raisonnement, retraitons l'exemple précédent avec un multiplexeur $4 \rightarrow 1$. Les variables A et B sont assimilées aux entrées de sélection. Pour obtenir le cablage de l'entrée E_0 du multiplexeur, la table de vérité initiale est réduite aux combinaisons où $AB = 00$. Soit $OUT = 0$ quelle que soit la valeur de C. Lorsque $AB = 00$, $E_0 = OUT$ donc $E_0 = 0\ \forall C$. En appliquant ce raisonnement aux trois entrées de multiplexage restantes, on obtient le cablage de la figure 2.2.

L'inconvénient majeur de cette approche réside dans le fait que le choix de l'affectation des variables d'entrées du système aux entrées de sélection du multiplexeur est aléatoire. Pour obtenir une solution optimale, il faudrait tester tous les choix possibles et ne retenir que la meilleure solution, méthode exhaustive qui peut-être très couteuse en temps.

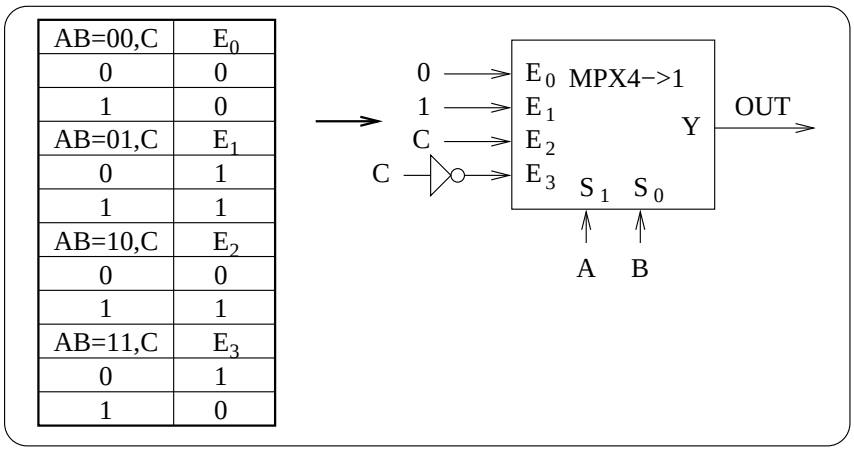


FIG. 2.2 – Synthèse indirecte à l'aide d'un multiplexeur

2.2.3 Multiplexeurs et circuits arithmétiques : les UALs

Les UALS (Unité arithmétique et logique) sont une des briques de base de tout microprocesseur actuel. Elles sont composées de 2^n circuits arithmétiques différents et de multiplexeurs à n bits de sélections qui permettent de transmettre en sortie, un seul des 2^n résultats calculés. Notons que le nombre de bits occupés par le résultat (par exemple 8 bits) conditionne le nombre de multiplexeurs nécessaires (ici 8) pour transmettre correctement ce résultat.

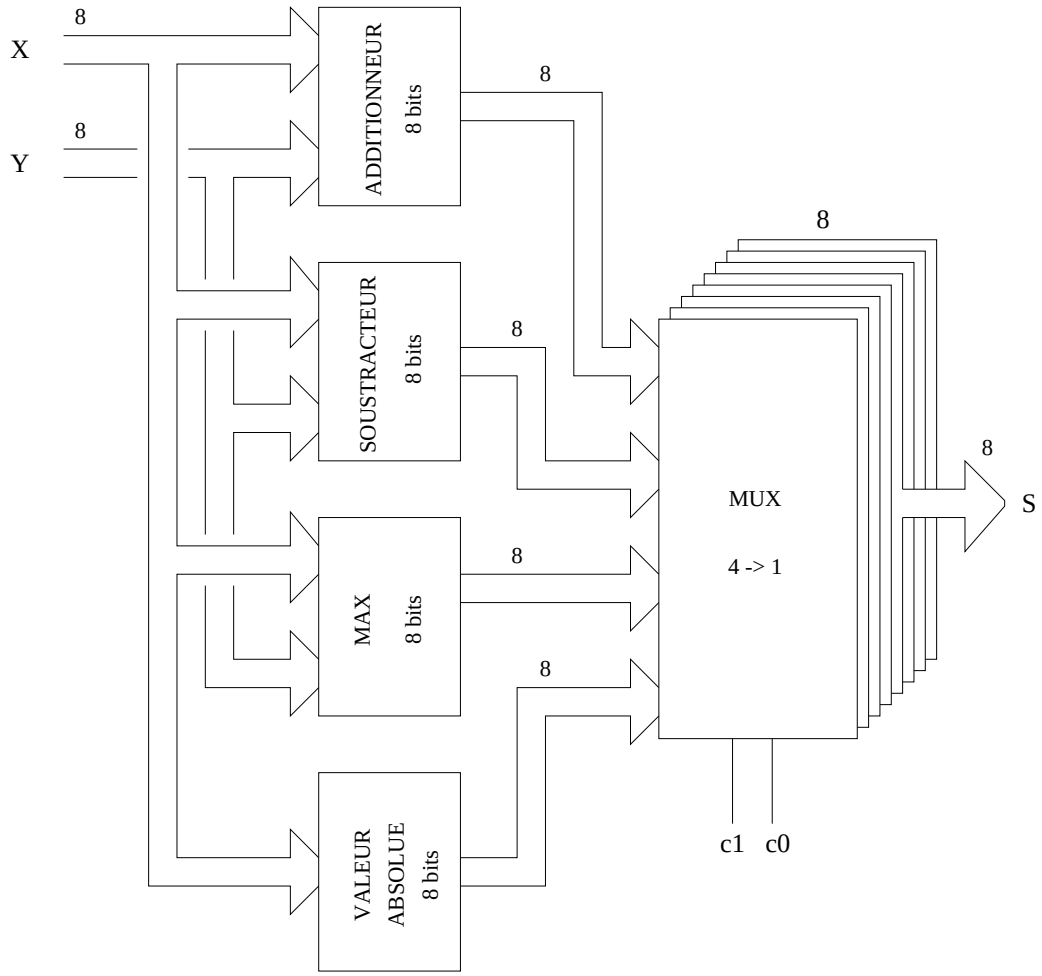


FIG. 2.3 – Exemple d'UAL 8 bits à quatre opérations

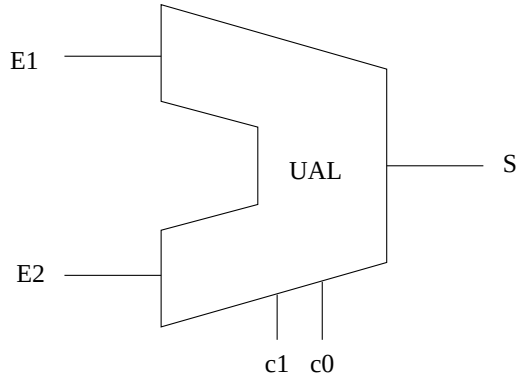


FIG. 2.4 – Symbole normalisé d'une UAL

2.3 Le démultiplexeur

2.3.1 Fonction démultiplexage

A la sortie d'un multiplexeur, il faut redistribuer les informations. Le procédé de transfert est donc basé sur le phénomène inverse du multiplexage. Le démultiplexeur permet de convertir des informations binaires séries en informations binaires parallèles. Il possède une entrée pour n sorties. La

différence entre le multiplexeur et le démultiplexeur réside dans le sens de circulation de l'information. Seule la sortie qui porte le bon numéro est connectée à l'entrée. Les autres sorties sont (suivant le modèle de démultiplexeur choisi) :

- déconnectées : on dit alors qu'elles sont en "haute impédance",
- ou alors fixées à une référence logique constante (0 ou 1).

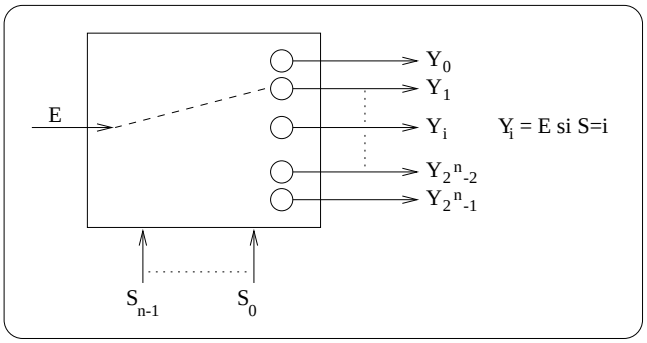


FIG. 2.5 – Schéma de principe du démultiplexeur

2.3.2 Fonction décodage

Un décodeur est un démultiplexeur particulier où l'entrée est fixée définitivement à "1"et les sorties non actives à 0. Le "1" d'entrée se retrouve sur la seule sortie active du décodeur.

Les décodeurs sont utilisés classiquement pour activer sélectivement un composant parmi n (par exemple un bloc mémoire particulier dans un ordinateur).

2.3.3 Générateur de mintermes

Le démultiplexeur (décodeur) constitue un générateur de mintermes. Un démultiplexeur comportant n entrées de commandes génère les 2^n mintermes des n variables d'entrées. Il suffit donc de faire un "OU" logique entre les sorties pour obtenir n'importe quelle fonction combinatoire. Cette propriété caractéristique sera d'ailleurs généralisée dans le prochain chapitre.

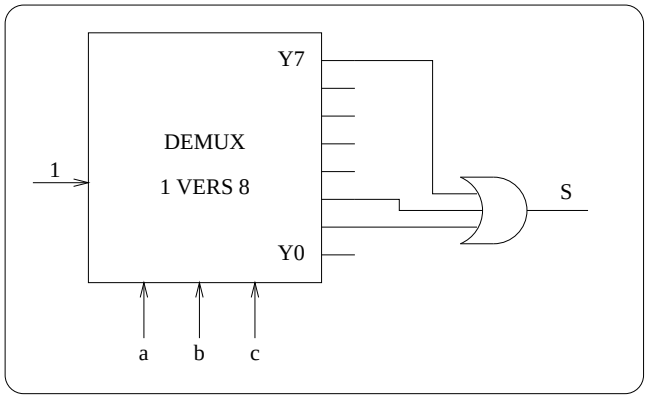


FIG. 2.6 – implémentation de $\overline{a}.\overline{b}.c + \overline{a}.b.\overline{c} + a.b.c$

Chapitre 3

Composants logiques programmables

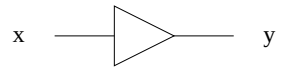
3.1 Notations complémentaires

L'étude des composants programmables, appele l'étude de nouvelles fonctions logiques :

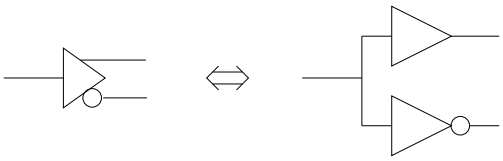
3.1.1 Amplificateur de ligne directe et inverse

Cette porte logique appelée "OUI" n'a aucun rôle logique vu sa table de vérité. Elle sert uniquement d'amplificateur de courant pour les portes suivantes (adaptation d'impédance). Elle se note comme suit et peut être associée à un inverseur :

x	y
0	0
1	1

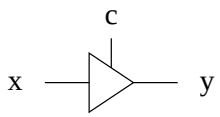


porte OUI

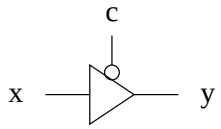


3.1.2 Trois-état (tri-state)

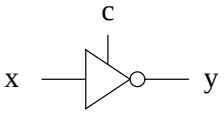
Un trois état est un composant logique qui permet de déconnecter physiquement une sortie sur commande (sortie en Haute Impédance).



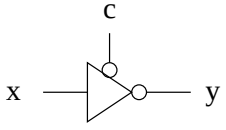
y=x si c=1
y en haute impédance si c=0



y=x si c=0
y en haute impédance si c=1



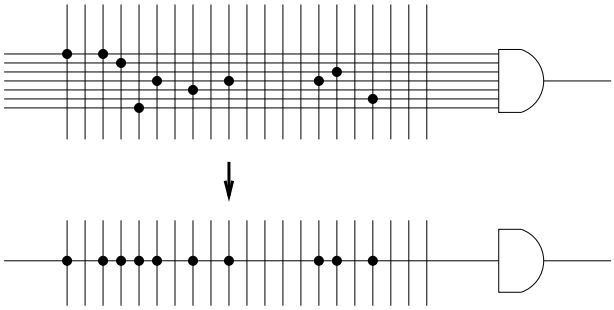
y=$\bar{x}$ si c=1
y en haute impédance si c=0



y=$\bar{x}$ si c=0
y en haute impédance si c=1

3.1.3 Représentation du ET logique

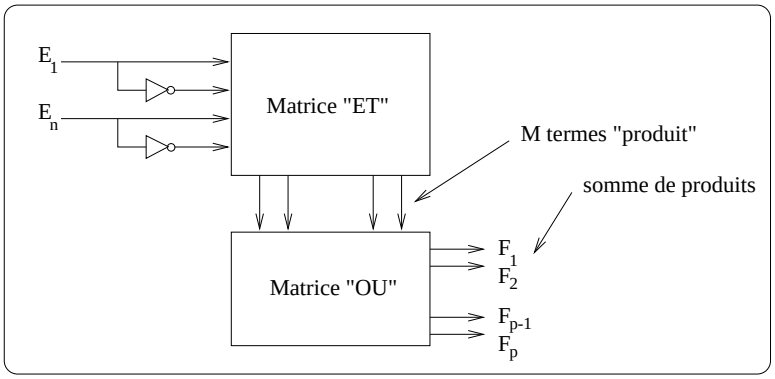
Sur les composants programmables, le nombre de ET peut être très grand. C'est pourquoi la convention suivante a été adoptée :



3.2 Principe des réseaux logiques programmables

Toute fonction logique peut se mettre sous la forme de somme de produits ou de produit de sommes. Il semble donc naturel d'utiliser une structure comportant deux ensembles fonctionnels :

- un ensemble d'opérateurs ET organisé sous forme d'une matrice qui génère les produits de variables d'entrées, éventuellement complémentées.
- un ensemble d'opérateurs OU, appelée matrice OU, qui somme les produits.



L'idée est de ne conserver que les produits et les sommes qui nous intéressent. Pour cela, le composant programmable prévoit au départ toutes les connexions possibles et laisse à l'utilisateur le soin de couper (griller les fusibles) les liaisons qui ne l'intéressent pas.

3.2.1 Classification des réseaux logiques combinatoires

Pour des réductions de surface et de complexité, les possibilités de programmation peuvent être réduites : le réseau Et seul ou le réseau OU seul est programmable. Ainsi, les réseaux logiques programmables recouvrent une grande gamme de produits qui différent selon la partie programmable :

- PROM : réseau ET programmé (décodeur), réseau OU programmable ,
- PAL : réseau ET programmable, réseau OU programmé, c'est à dire que la structure OU est définie,
- PLA : réseau ET et OU programmable.

Le schéma qui suit montre l'architecture interne d'un PAL comportant au plus 16 entrées et 8 sorties.

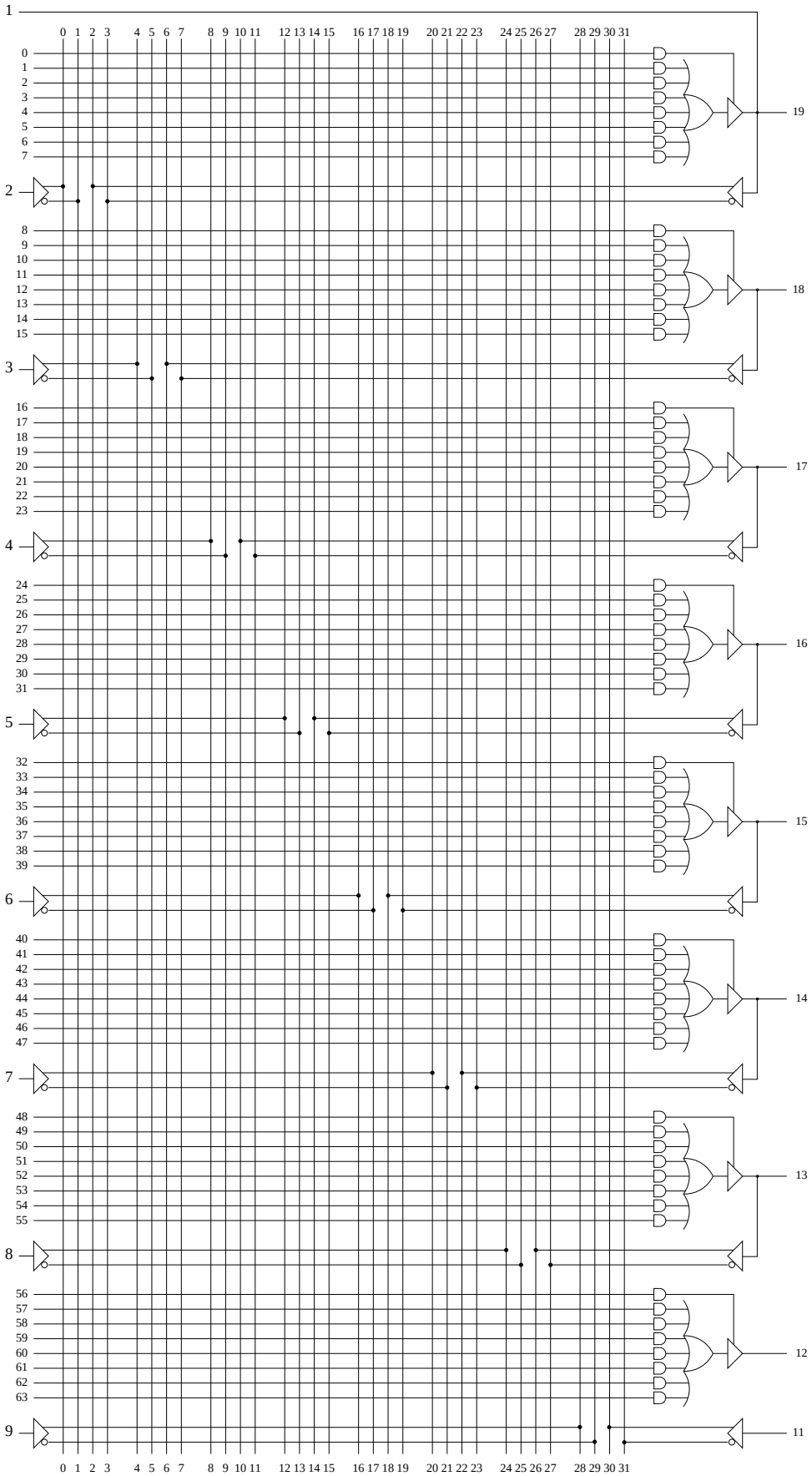


FIG. 3.1 – vue interne du PAL16H8 (document texas)

3.2.2 rebouclage

La grande majorité des composants programmables actuels permettent de réinjecter les sorties dans la matrice ET. Ceci a deux conséquences principales :

- Une fonction logique peut etre évaluée à partir d'une autre sans avoir à refaire une seconde fois les calculs.

- Si la sortie est configurée en haute impédance, elle peut servir alors d'entrée supplémentaire au système. C'est pour cette raison d'ailleurs que le schéma précédent peut accepter au plus 16 entrées.

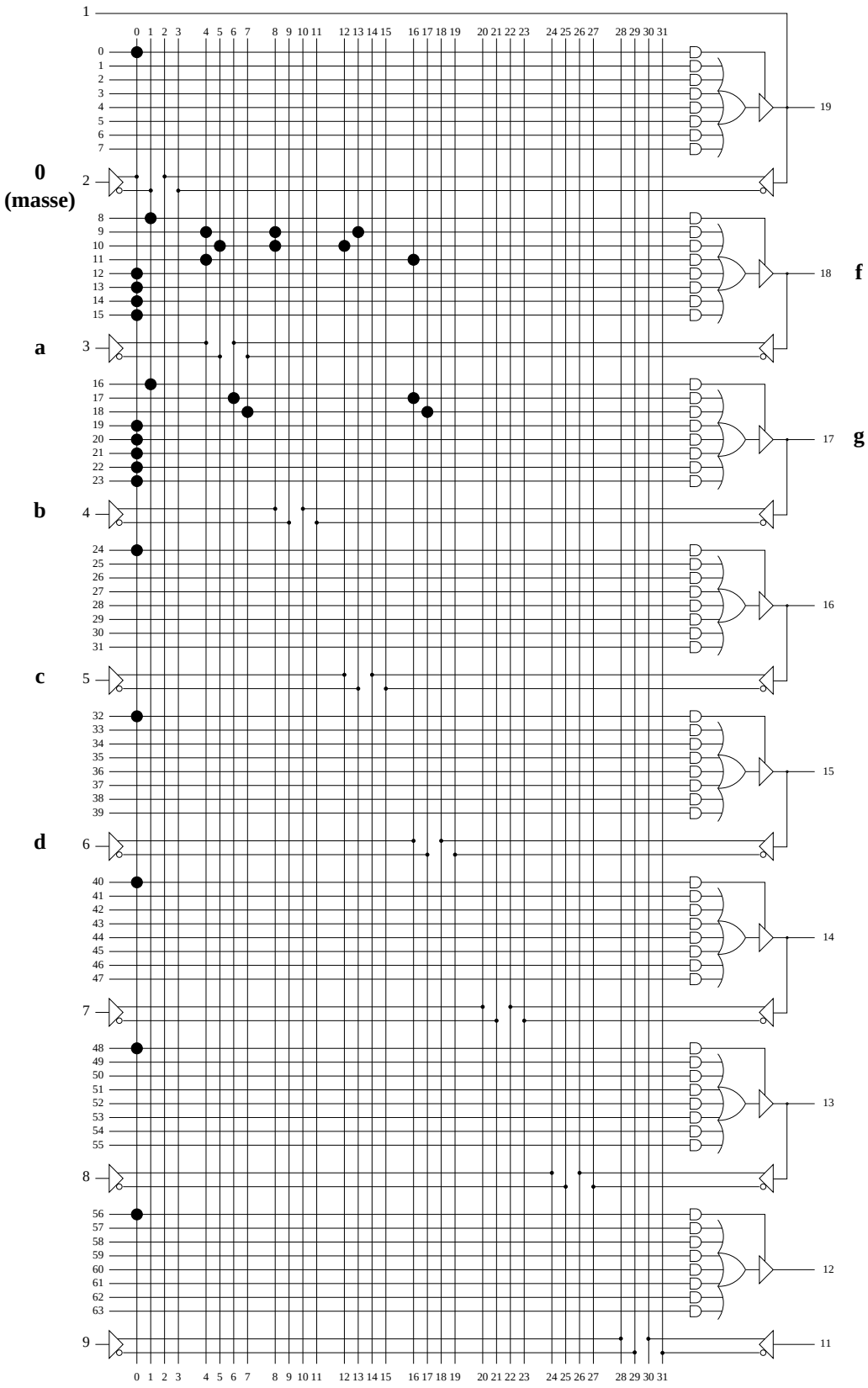
3.3 Exemple de programmation

Sur l'exemple de la page suivante, les points représentent les fusibles qui ont été conservé. De cette manière, deux fonctions ont été implanté[1] et toutes les sorties non utilisée ont éte mise en haute impédance pour éviter de griller le composant accidentellement :

- $f(a,b,c,d) = a.b.\overline{c} + \overline{a}.b.c + a.d$

- $g(a,b,c,d) = f(a,b,c,d).d + \overline{f(a,b,c,d)}.\overline{d}$

[1] Dans le jargon on dit souvent "mappé" de l'anglais map : carte


0 (masse)
a
b
c
d
f
g

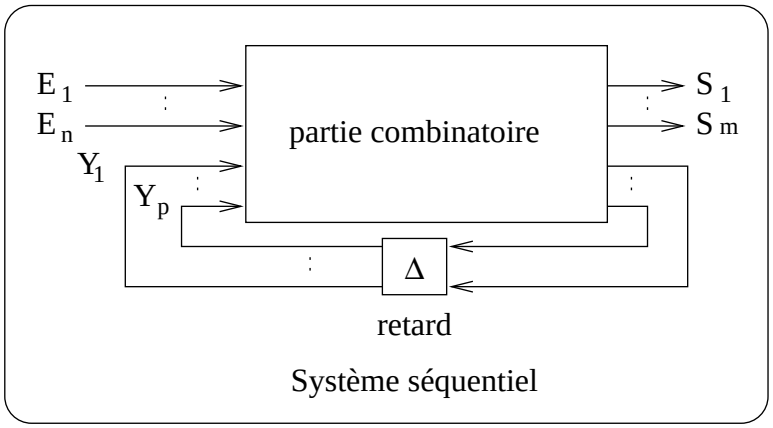
Chapitre 4

Introduction aux systèmes séquentiels : les bascules

4.1 Introduction

Nous avons étudié jusqu'ici des réseaux que nous avons appelés combinatoires, c'est à dire qui associent à toute combinaison binaire d'entrée une combinaison de sortie qui est unique. De ce type de fonctionnement il résulte en particulier que chaque fois qu'on présentera une combinaison C donnée à l'entrée, à des instants distincts, on verra apparaître la même combinaison à la sortie. La correspondance entrée-sortie établie par un tel réseau est donc ponctuelle, elle est indifférente à ce qui s'est passé entre les différents instants où on a appliqué la combinaison C : elle ne fait intervenir aucun phénomène de mémoire de ce qui s'est passé avant. La différence essentielle entre un système combinatoire et un système séquentiel est que l'état du second ne dépend pas uniquement de la combinaison d'entrée à un moment donné mais de la séquence des combinaisons précédentes et de son état initial. Un circuit séquentiel possède donc un *état interne* qui caractérise le chemin chronologique qu'il a parcouru depuis sa création.

En conséquence à une combinaison des variables d'entrées ($e_1 ... e_n$) correspondent plusieurs états possibles du système. Pour différencier ces états, il faut introduire un certain nombre de variables supplémentaires, appelées variables internes ($y_1 ... y_p$), l'état du système étant fixé par la combinaison ($e_1 ... e_n, y_1 ... y_p$). Un système séquentiel peut alors être considéré comme constitué d'un circuit combinatoire ayant pour entrée $e_1 ... e_n$, $y_1 ... y_p$, ils apparaissent ainsi comme des systèmes bouclés.



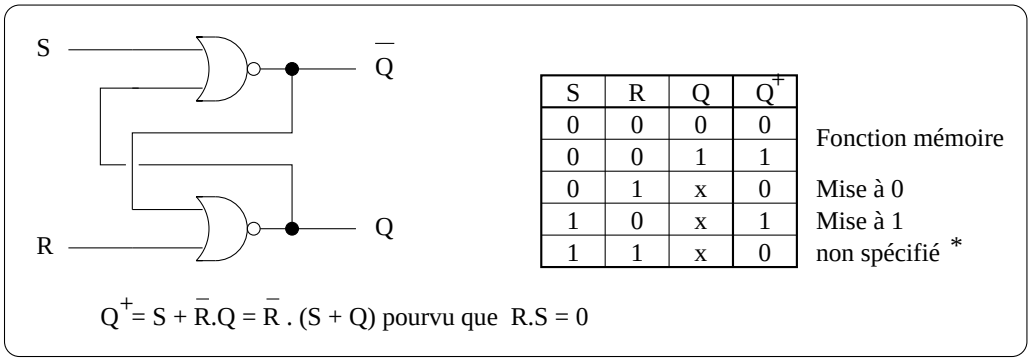
Suivant les conditions de changement des variables de sortie, on distingue deux types de système séquentiel :

– les systèmes séquentiels asynchrones pour lesquels l'état des sorties évolue spontanément à la suite d'un changement de configuration des variables d'entrée.
– les systèmes séquentiels synchrones pour lesquels l'évolution des sorties est assujettie à une commande spécifique généralement appelée horloge. Le système est alors synchronisé sur le signal d'horloge.

4.2 Bascules asynchrones

4.2.1 La bascule SR

La bascule SR dispose de deux entrées, S et R, l'une pour la mise à 1 de la sortie Q et l'autre pour la mise à zéro.



S	R	Q	Q^+	
0	0	0	0	Fonction mémoire
0	0	1	1	
0	1	x	0	Mise à 0
1	0	x	1	Mise à 1
1	1	x	0	non spécifié *

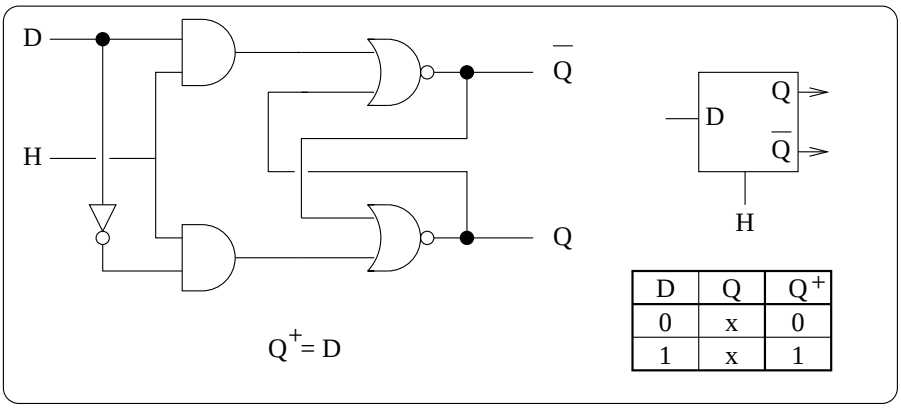
$Q^+ = S + \overline{R}.Q = \overline{R} . (S + Q)$ pourvu que $R.S = 0$

* n'appartient pas au domaine de spécification de la bascule

En appelant Q la valeur de la sortie à l'instant t, R et S provoqueront au même instant une évolution de la sortie vers une valeur que l'on nomme Q^+.

4.2.2 La bascule D active sur niveau (Latch D)

Cette bascule dispose d'une seule entrée de donnée D. Le signal de synchronisation est actif sur niveau H (en anglais Latch D). Lorsque le signal de synchronisation est actif, la sortie recopie l'entrée D sur la sortie Q, sinon la sortie conserve (mémorise) la dernière valeur de D où le signal de synchronisation était actif.

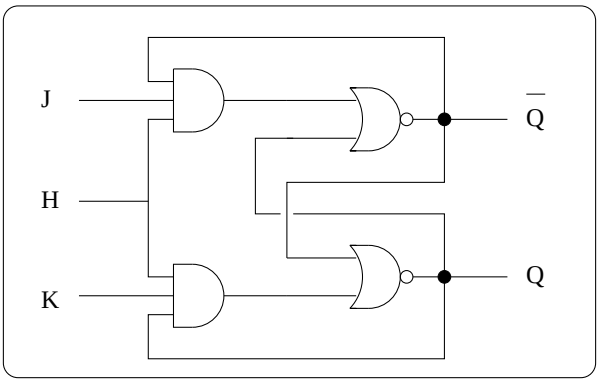


D	Q	Q^+
0	x	0
1	x	1

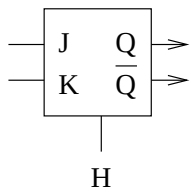
$Q^+ = D$

4.2.3 La bascule JK active sur niveau

C'est une bascule disposant de deux entrées, respectivement appelées J et K. Comme pour la bascule SR, l'entrée J sert pour la mise à 1 et l'entrée K pour la mise à 0. Outre la synchronisation par un niveau haut sur H, la différence entre la bascule JK et la bascule SR réside dans le fait qu'il n'y a plus d'état interdit (non spécifié) pour les entrées.



Voici sa table de vérité :



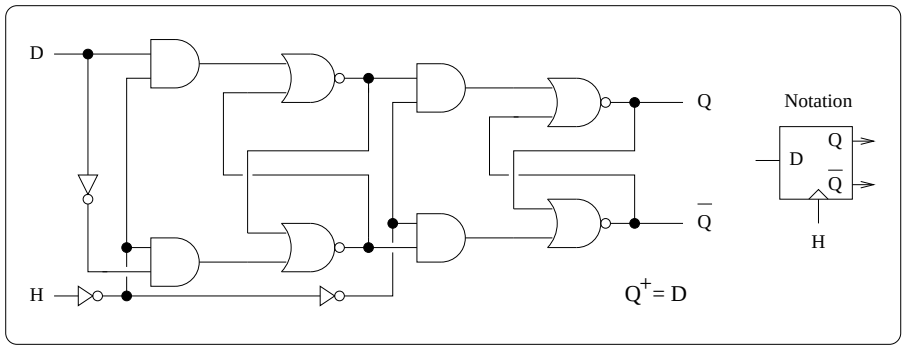
$Q^+ = J.\overline{Q} + \overline{K}.Q$

J	K	Q	Q^+	
0	0	0	0	
0	0	1	1	Fonction mémoire
0	1	0	0	
0	1	1	0	Mise à 0
1	0	0	1	
1	0	1	1	Mise à 1
1	1	0	1	
1	1	1	0	oscillation

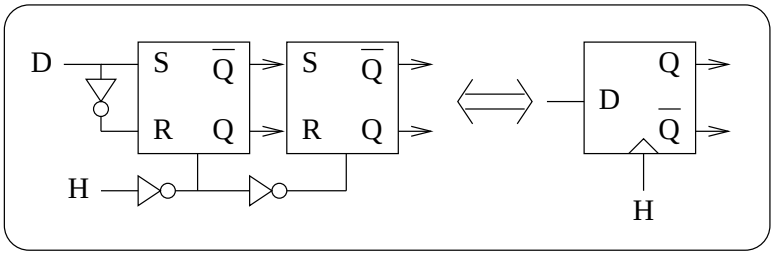
4.3 Bascules synchrones

4.3.1 La bascule D active sur front (Flip-Flop D ou Edge Triggered D)

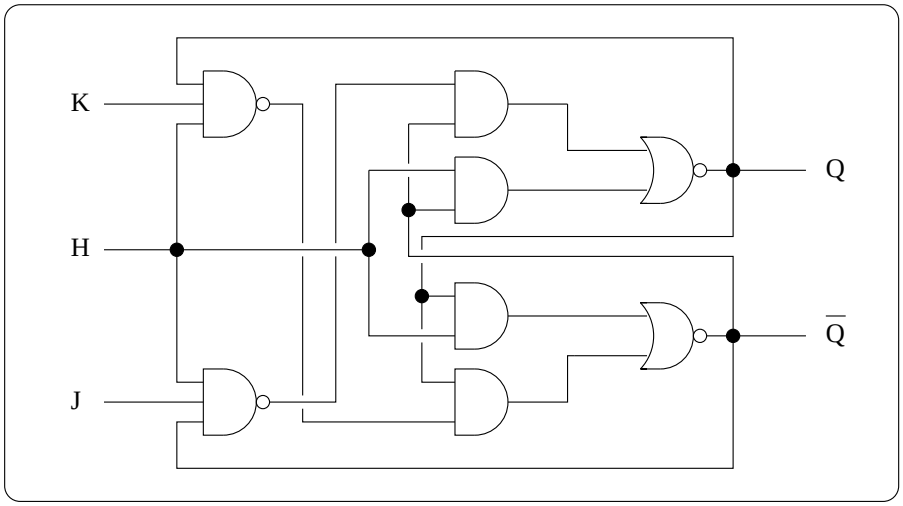
Comme précédement, cette bascule dispose d'une seule entrée D et d'une sortie Q, par contre le signal de synchronisation H est actif sur un front (en anglais Edge Triggered). La figure suivante, montre une implémentation possible d'une bascule D active sur front montant :



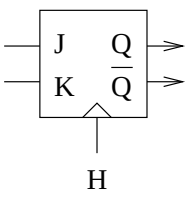
Cette classe de bascule est également appelée maître-esclave (en anglais Flip-Flop) car elle est constituée de deux bascules dont l'une pilote l'autre :



4.3.2 La bascule JK active sur front (Edge JK)



Ormis son mode de déclenchement sur front montant, sa table de vérité est conforme à celle de la bascule JK asynchrone :



$Q^+ = J.\overline{Q} + \overline{K}.Q$

J	K	Q	Q^+	
0	0	0	0	
0	0	1	1	Fonction mémoire
0	1	0	0	
0	1	1	0	Mise à 0
1	0	0	1	
1	0	1	1	Mise à 1
1	1	0	1	
1	1	1	0	

sur front montant de H

Chapitre 5

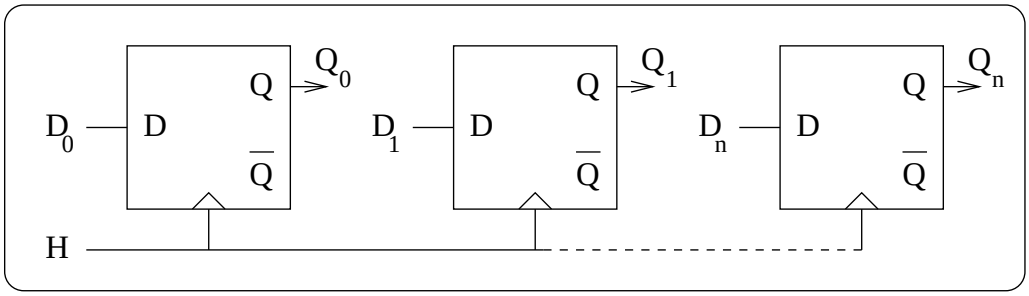
Les registres

5.1 Introduction

La bascule nous a permis de mettre en mémoire une information binaire élémentaire. Le registre composé de plusieurs bascules permet de conserver une information de plusieurs digits. De plus l'interconnexion entre les bascules permet certaines manipulations de l'information stockée.

5.2 Fonctionnement

Un registre sert à mémoriser un mot ou nombre binaire. Le schéma d'un tel système comporte autant de bascules type D que d'éléments binaires à mémoriser. Toutes les bascules sont commandées par le même signal d'horloge.



Moyennant une interconnexion entre les cellules, le registre précédent devient capable d'opérer une translation des chiffres du nombre initialement stocké. Le déplacement s'effectue soit vers la droite soit vers la gauche. Le registre est alors appelé registre à décalage. De nombreuses applications résultent de cette possibilité de décalage :

- conversion série-parallèle d'une information numérique,
- division et multiplication par une puissance de 2,
- la ligne à retard.

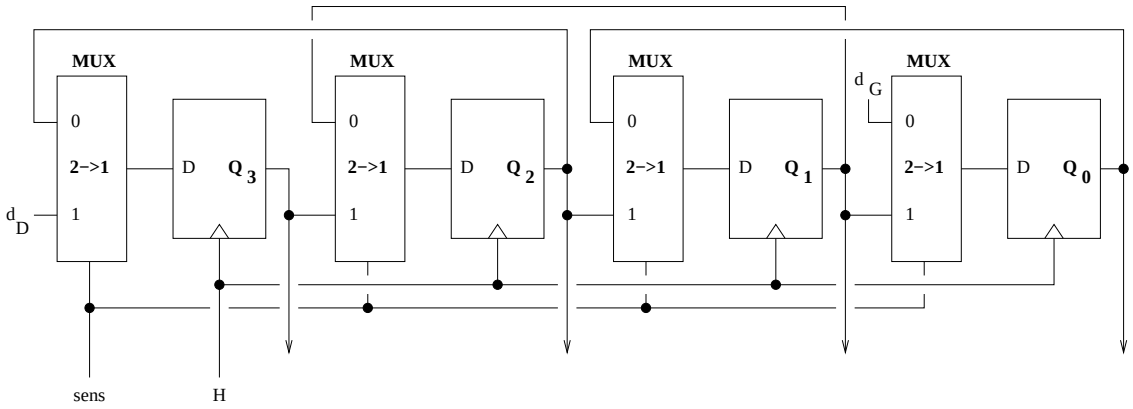
5.3 Registres et multiplexeurs

Rappelons qu'un multiplexeur est d'abors un aiguillage élémentaire de données. Par cette caractéristique, nous pouvons donc changer dynamiquement les interconnexions entre les registres et ainsi modifier le comportement global du système.

Le schéma 5.3 représente un tel système :

Lorsque l'entrée de sélection $sens$ est positionnée à "1", la sortie du registre n est reliée à l'entrée du registre $n-1$ et nous obtenons ainsi un décaleur à droite.

Par contre le positionnement de $sens$ à "0" permet de connecter la sortie du registre n à l'entrée du registre $n+1$ et la donnée se retrouve décalée à gauche.



Bien-sur, cette architecture peut être améliorée en prenant des multiplexeurs plus complexes !

Chapitre 6

Analyse et synthèse de compteurs synchrones

6.1 Introduction

Les compteurs sont des éléments essentiels de logique séquentielle ; ils permettent en effet d'établir une relation d'ordre de succession d'événements. L'état du compteur est défini par le nombre binaire formé par l'ensemble de sorties des bascules.

On distingue deux grandes familles de compteurs qualifiées de "*synchrone*" et "*d'asynchrone*". Dans les compteurs synchrones, toutes les bascules sont synchronisées sur une seule et même horloge H. Dans les compteurs asynchrones, chaque bascule génère un nouveau signal d'horloge pour la suivante. Le schéma qui suit, montre un tel système (décompteur asynchrone cyclique de 7 à 0).

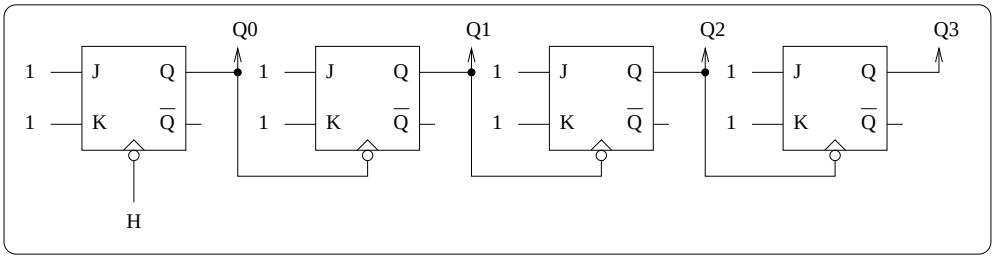


FIG. 6.1 – décompteur asynchrone

Dans le cas général, on évitera de multiplier les horloges et de jouer sur les temps de propagation car ces systèmes deviennent très rapidement complexes à concevoir. On leur préfèrera les compteurs synchrones beaucoup moins sensibles aux transitoires.

Ces compteurs synchrones peuvent toujours se représenter sous forme de deux sous-ensembles : une mémoire (registre) dont l'élément de base est la bascule et une fonction f de calcul du nombre suivant (l'état futur : Q^+).

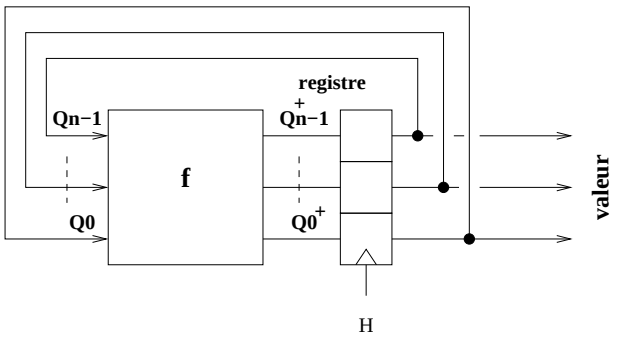


FIG. 6.2 – Schématisation d'un compteur synchrone

6.2 Synthèse de compteurs synchrones

6.2.1 Méthode de synthèse directe

L'objectif de ce travail de synthèse est de déterminer les équations logiques de f afin de conditionner les évolutions des bascules lors de la prochaine impulsion d'horloge. Le problème posé est donc le suivant : à un instant t, quelles sont les valeurs à appliquer aux entrées pour qu'à l'impulsion d'horloge suivante la sortie prenne la valeur imposée par le cycle de comptage désiré ?

Dans un premier temps, on établit la table de vérité qui traduit la succession des états du compteur. Cette table est ici appellée *table d'évolution*. Les entrées de la table représentent l'état présent du compteur alors que les sorties représentent l'état futur. Prenons à titre d'exemple un compteur binaire 2 bits :

Q_2	Q_1	${Q_2}^+$	${Q_1}^+$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

De cette table d'évolution, il est aisé d'en déduire les équations logiques des variables de sortie, dans le cas de notre exemple,

$$\begin{aligned} {Q_1}^+ &= \overline{Q_1} \\ {Q_2}^+ &= Q_1.\overline{Q_2} + \overline{Q_1}.Q_2 \end{aligned}$$

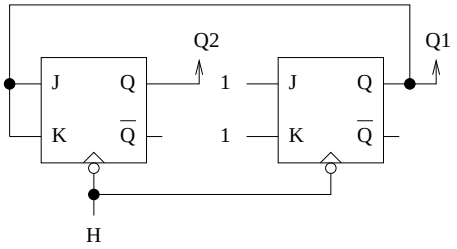
Pour obtenir les expressions des entrées des bascules, il faut, dans un second temps, procéder par identification avec l'expression de la bascule considérée. Par exemple, considérons la bascule JK dont l'équation de sortie est $Q^+ = J.\overline{Q} + \overline{K}.Q$.

J représente alors le terme mis en facteur par rapport à $\overline{Q}$ et K par rapport à Q. La seule précaution à prendre consiste à exprimer Q^+ en fonction de Q et $\overline{Q}$ même si cela ne représente pas l'expression minimale de Q^+.

Reprenons l'exemple du compteur binaire deux bits et supposons que notre réalisation ne comporte que des bascules JK, l'identification nous donne :

$$\begin{aligned} J1 &= 1, \\ K1 &= 1, \\ J2 &= Q1, \\ K2 &= Q1. \end{aligned}$$

Ce qui correspond au schéma suivant :



Le principal inconvénient de cette méthode réside dans le fait que l'équation minimale des variables des sorties des bascules ne permet pas toujours une identification aisée avec l'expression des entrées.

6.2.2 Méthode de Marcus

L'objectif de la méthode de Marcus est de déterminer la table de vérité des entrées des bascules afin d'obtenir leur expression minimale.

Dans un premier temps, on établit comme précédemment la table d'évolution du compteur. A partir de cette dernière et de la table de fonctionnement de la bascule utilisée, on construit un autre tableau où les sorties représentent les valeurs à appliquer aux entrées des bascules. Le rôle de la table de fonctionnement est de donner la valeur des entrées à appliquer pour les quatre cas possibles d'évolutions de la bascule considérée.

Dans le cas d'une bascule JK, cette table est :

Q	Q^+	J	K
0	0	0	$\emptyset$
0	1	1	$\emptyset$
1	0	$\emptyset$	1
1	1	$\emptyset$	0

Si nous reprenons l'exemple du compteur binaire 2 bits, nous obtenons la table d'évolution qui suit :

Q_2	Q_1	J_2	K_2	J_1	K_1
0	0	0	$\emptyset$	1	$\emptyset$
0	1	1	$\emptyset$	$\emptyset$	1
1	0	$\emptyset$	0	1	$\emptyset$
1	1	$\emptyset$	1	$\emptyset$	1

Cette table nous permet d'en déduire aisément les équations logiques des entrées des bascules (ici les mêmes que précédement). En comparaison avec la première méthode, il y a deux fois plus de fonctions à simplifier mais on est assuré d'obtenir des équations plus simple que l'identification.

6.3 Compteurs incomplets et cycle parasite

Dans le cas des compteurs incomplets, les deux méthodes (directe et Marcus) peuvent être utilisées indistinctement. La seule différence porte sur les états futurs non spécifiés qui seront notés comme *indifférent* pour obtimiser les équations.

Il faudra tout de même vérifier *a posteriori* que les états futurs induisent bien une séquence de comptage qui retourne dans le cycle voulu au bout d'un nombre fini de fronts d'horloge. Dans le cas contraire, nous seront en présence d'un cycle parasite ou d'un état "puit".

Ce problème se rencontre également sur des compteurs déjà existant :

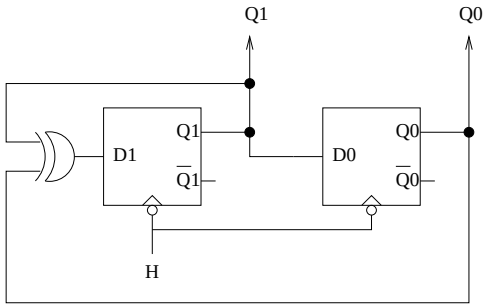


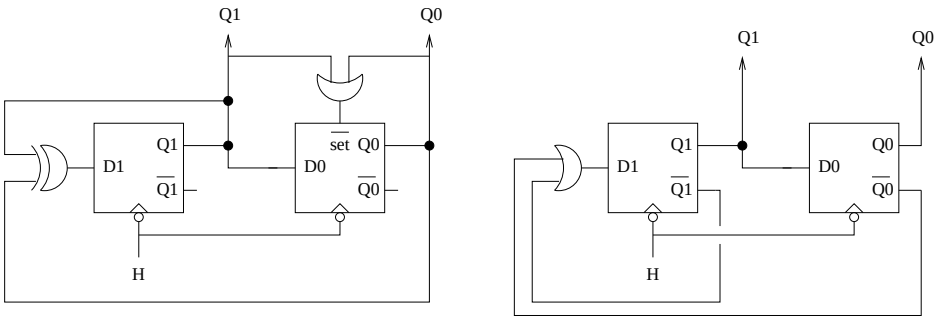
FIG. 6.3 – Compteur de 1 à 3 avec cycle parasite (état puit)

Sur la figure 6.3, les équations de changement d'état donnent : $Q1^+ = Q1 \oplus Q0$ et $Q0^+ = Q1$, soit la table de changement détat :

$Q1$	$Q0$	$Q1^+$	$Q0^+$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Nous retrouvons bien le cycle 1,2,3. Par contre 0 conduit à 0. A la mise sous tension, nous risquons donc de voir notre compteur bloqué sur 0.

Pour palier ce probleme, il faut "casser" ce cycle ($00 \rightarrow 01$). La solution la plus rapide consiste à ajouter une "rustine" sur le compteur vers le `set`, la plus propre à refaire la synthèse !



''rustine'' vers le set **nouvelle synthèse**

Chapitre 7

Les séquenceurs

7.1 Constitution

Un séquenceur est à la base un compteur, qui active à chaque coup d'horloge, zero, une ou plusieurs actions particulières. Pour simplifier la réalisation technique, on utilise classiquement un décodeur dont chaque ligne est activée en séquence. Ces lignes viennent ensuite piloter des portes logiques OU qui fond l'union de tous les états où l'action considérée doit être faîte.

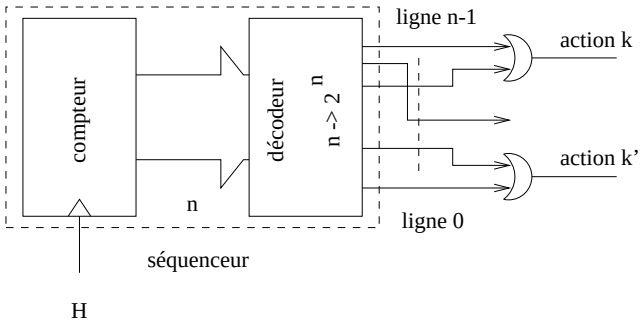


FIG. 7.1 – Structure d'un séquenceur câblé

Le chronogramme suivant montre l'incidence du compteur sur l'activité des lignes.

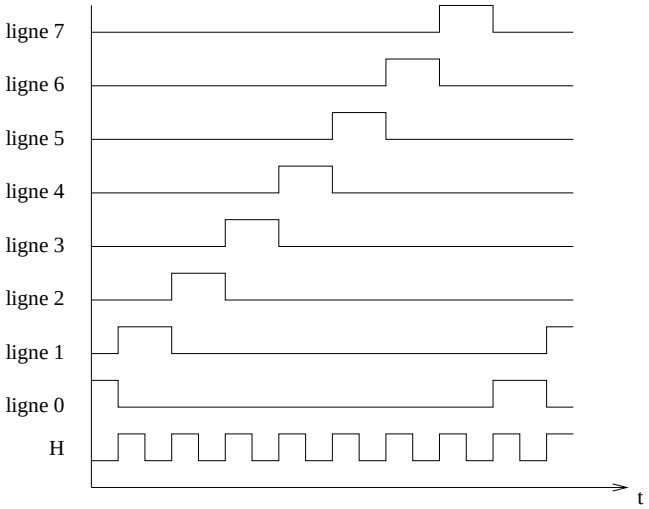


FIG. 7.2 – Chronogramme de base d'un séquenceur à 8 états

7.2 Applications

7.2.1 Feux de croisement

Le premier exemple simple illustre une manière possible de générer la séquence de feux de croisement :

feu 1	feu piétons 1	feu 2	feu piétons 2
R		V	r
R	r	V	r
R	r	O	r
V	r	R	
V	r	R	r
O	r	R	r

Il nécessite un séquenceur à 6 états donc un compteur à 6 états (par exemple de 0 à 5). Chaque ligne du séquenceur correspond à une ligne de la table d'évolution. Ainsi la première ligne doit allumer le feu 1 Rouge, le feu 2 Vert, le feu 2 piéton et eteindre le feu 1 piéton ...

D'ou le schéma suivant :

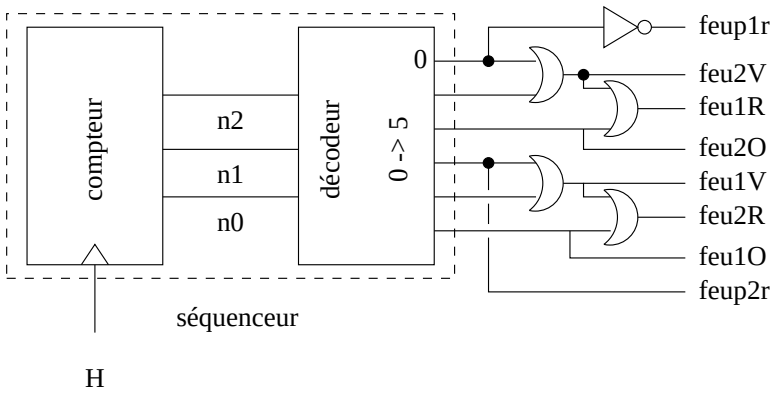


FIG. 7.3 – Gestion de feux de croisement

Nota : Les temps de chaque cycle sont ici constants. Ce résultat se généralise facilement en considérant un temps de base (horloge) PGCD de chaque cycle.

7.2.2 Multiplication non signée

Considérons le produit non signé de A et de B codés sur 4 bits. L'idée sous-jacente est de cumuler dans un registre la somme intermédiaire. Chaque ligne à sommer est en fait soit 0, soit A décalé à gauche. En effet A fois 0 rend 0 et A fois 1 rend A.

A			1	0	1	0	
B		×	1	0	1	1	
			1	0	1	0	
		1	0	1	0		
	0	0	0	0			
1	0	1	0				
1	1	0	1	1	1	0	

Pour sélectionner successivement tous les bits de B, une solution consiste à prendre le bit de poid faible de B et à décaler B à droite. Nous aurons donc besoin de deux registres à décalage (pour A : décalage à gauche, pour B : décalage à droite). Ces registres ont deux commandes : chargement et décalage (cf ch.5) synchronisés sur horloge.

Le schéma ci-dessous montre le multiplieur complet. On remarque la présence de portes ET qui forcent l'entrée de l'additionneur à 0 suivant le bit de poids faible de A.

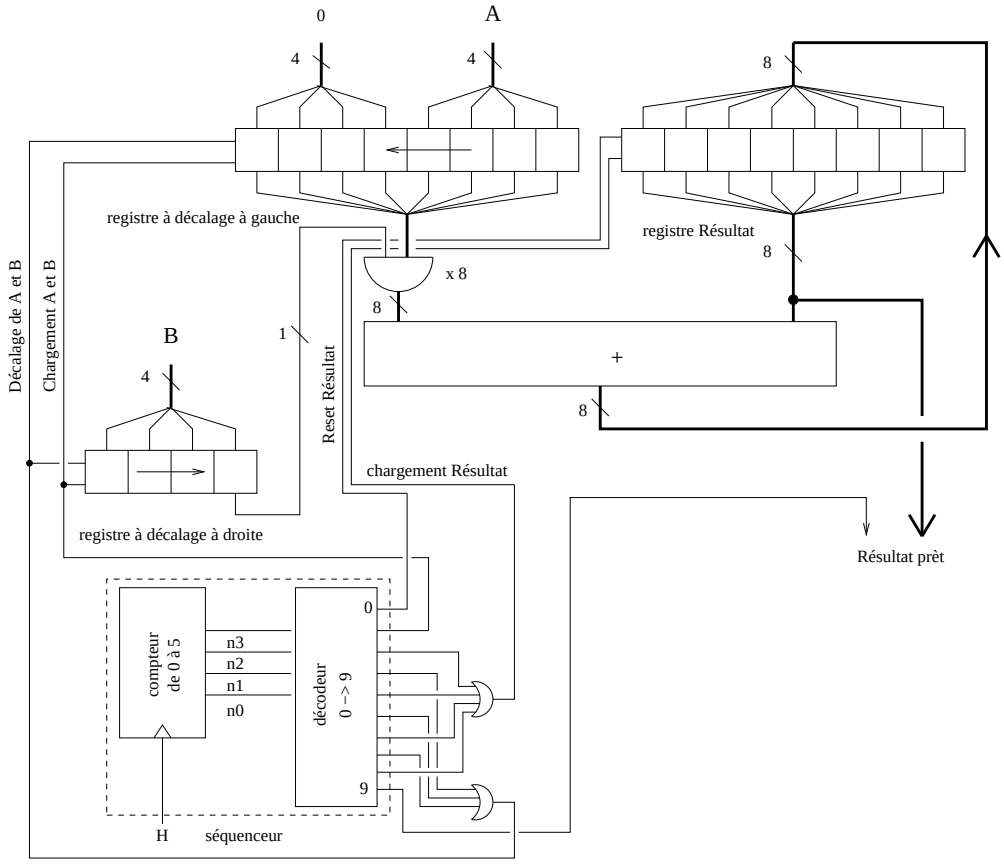


FIG. 7.4 – Multiplieur

Le séquenceur est cablé pour générer la séquence suivante :

- Reset Résultat
- Chargements A et B
- Chargement Résultat
- Décalages A et B

- Chargement Résultat
- Décalages A et B
- Chargement Résultat
- Décalages A et B
- Chargement Résultat
- Résultat prèt

On peut remarquer que ce multiplieur n'est pas très performant car il additionne parfois des "0". En fait nous aurions eu besoin d'un séquenceur conditionnel, mais ceci fera l'objet d'un cours de l'année prochaine ...