Corrigé TD d'électronique n°2

Logique combinatoire (2)

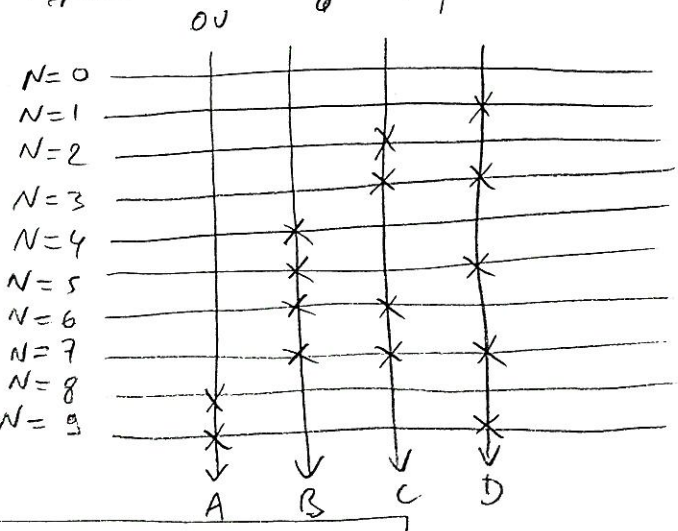
TD2 — Logique Combin.2

1

Table de codage

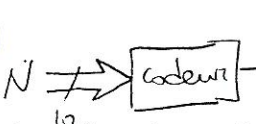
N	ABCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Représentation symbolique



1 seule entrée active à la fois ≡ Codeur ≡ Matrice OU

2



≡ de l'exercice précédent (1)

↑ 1 seule entrée active (1 parmi 10 ici)

3

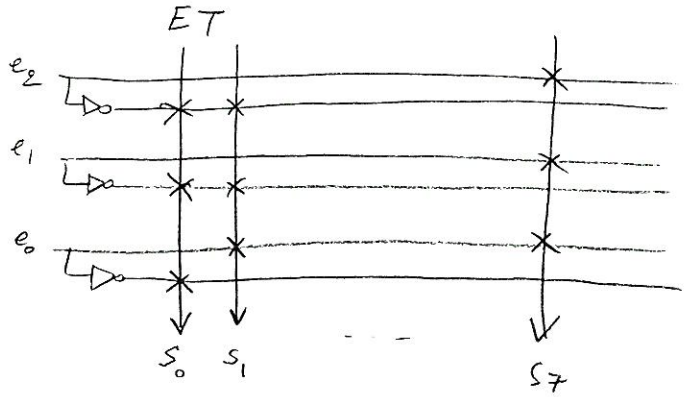
1 seule sortie active à la fois ≡ Décodeur ≡ Matrice ET

Table de Décodage

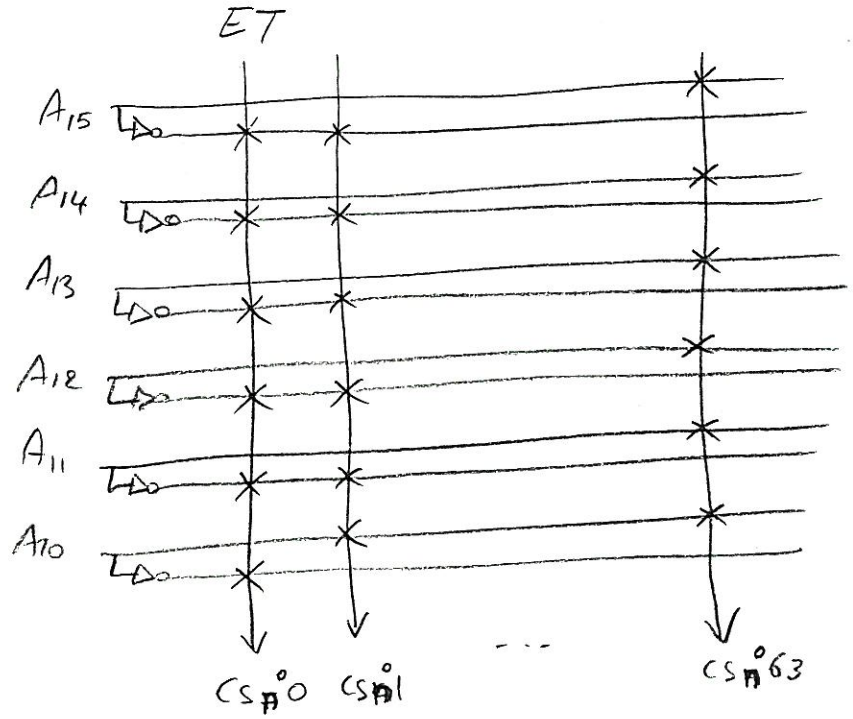
e_2 e_1 e_0	S_i
0 0 0	$S_0 = 1$ ($S_i = 0$ de $i = 1$ à 7)
0 0 1	$S_1 = 1$ ($S_0 = 0$; $S_i = 0$ de $i = 2$ à 7)
0 1 0	$S_2 = 1$ ($S_0 = S_1 = 0$; $S_i = 0$ de $i = 3$ à 7)
0 1 1	$S_3 = 1$
1 0 0	$S_4 = 1$
1 0 1	$S_5 = 1$
1 1 0	$S_6 = 1$
1 1 1	$S_7 = 1$

etc...

Représentation symbolique



4



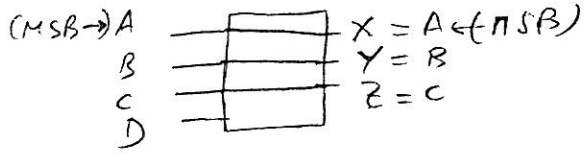
5

Multiplier par 2 ≡ Décaler le mot binaire vers la gauche
Diviser par 2 ≡ ——— droite

car. N : [... | b_i | ...] = ... + $b_i \cdot 2^i$ + ...

$N \times 2 \longrightarrow b_i \cdot 2^{i+1} = 2 \times b_i \cdot 2^i$ ($b_i \cdot 2^{i+1}$, 2^i)

Le bit introduit par décalage étant à 0. ex : 2 sur 4 bits = 0010 → (b_i) 4 = 0100 et 1 = 0...
→ Pas besoin de faire la table de transcodage, il suffit de prendre les bits décalés de 1 cran :



6

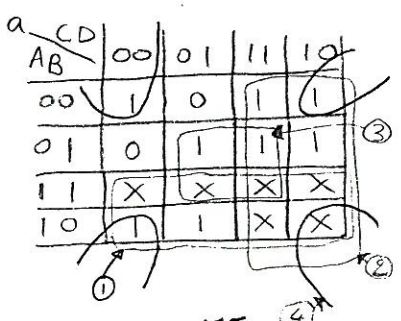
a) idem que ~~l'ex.~~ 7. mais les 6 dernières lignes sont indéfinies (X) pour a b c d e f g

	A B C D	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
	idem	idem
9	1 0 0 1	1 1 1 1 0 1 1
A	1 0 1 0	X X X X X X X
B	1 0 1 1	X X X X X X X
C	1 1 0 0	X X X X X X X
D	1 1 0 1	X X X X X X X
E	1 1 1 0	X X X X X X X
F	1 1 1 1	X X X X X X X

(6) ~~abis~~ abis).

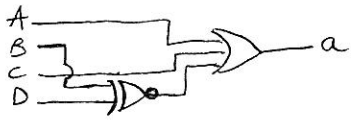
N	N(BCD) A	B	C	D	N(7 segments) a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
A	1	0	1	0	1	1	1	0	1	1	1
B	1	0	1	1	0	0	1	1	1	1	1
C	1	1	0	0	1	0	0	1	1	1	0
D	1	1	0	1	0	1	1	1	1	0	1
E	1	1	1	0	1	0	0	1	1	1	1
F	1	1	1	1	1	0	0	0	1	1	1

b)

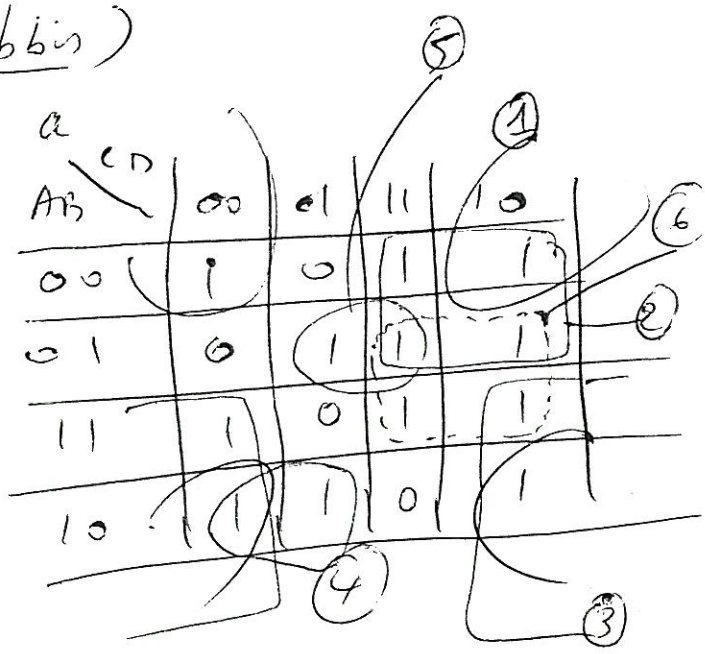


$a = A + C + BD + \overline{B}\,\overline{D} = A + C + \overline{B \oplus D}$

① ② ③ ④



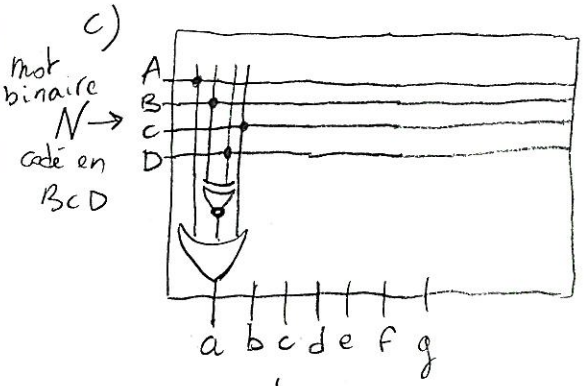
bbis)

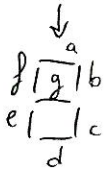


$a = \overline{B}\,\overline{D} + \overline{A}C + A\overline{D} + A\overline{B}\overline{C} + \overline{A}BD + BC$

0 ① ② ③ ④ ⑤ ⑥

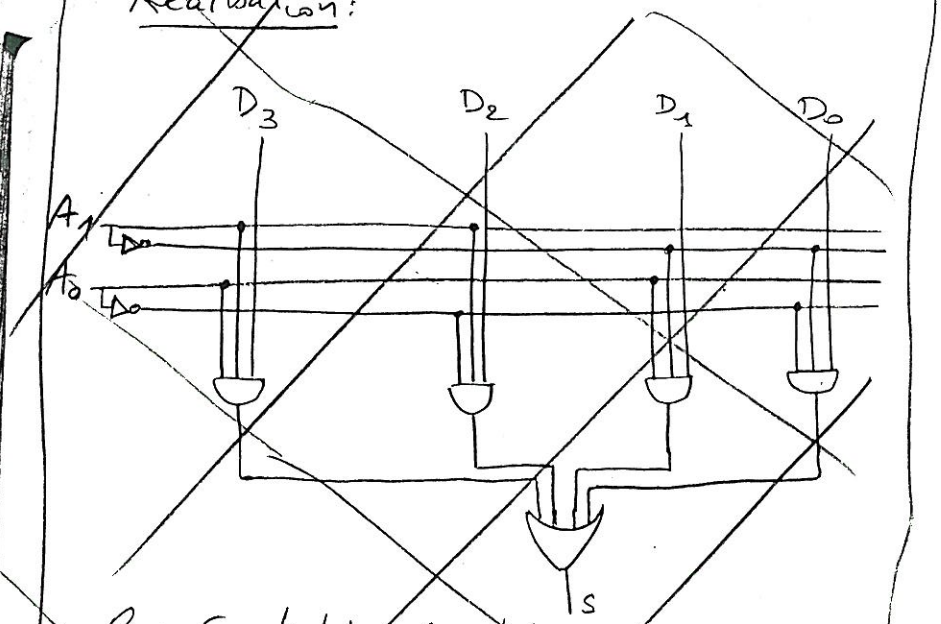
c)



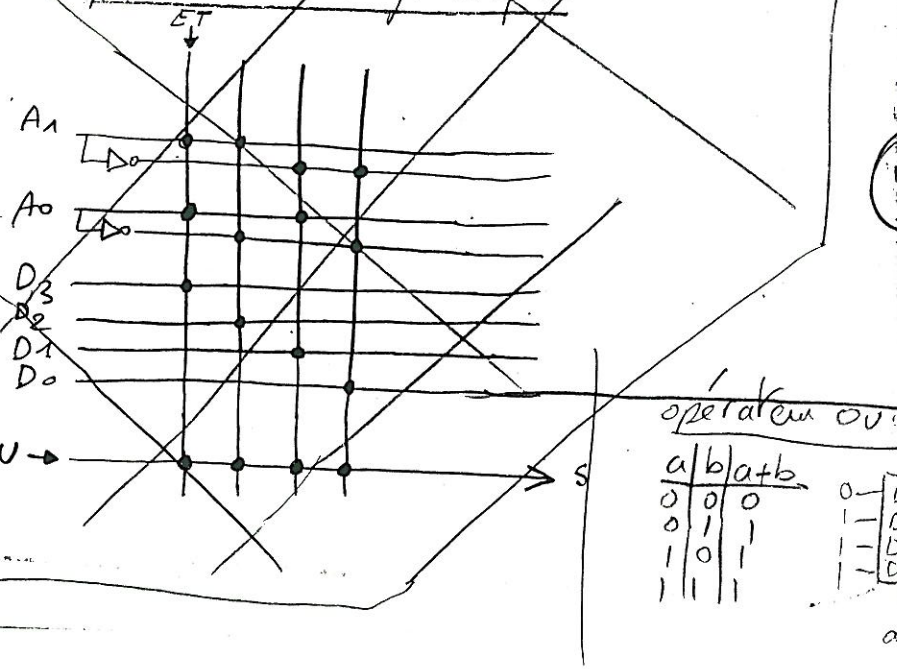


(7)

4.4.2 Réalisation :



Représentation symbolique



opérateur OU :

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

8

9

12.4.4.3 Applications :

12.4.4.3.1 Sélection d'un bit parmi plusieurs bits

mais aussi :

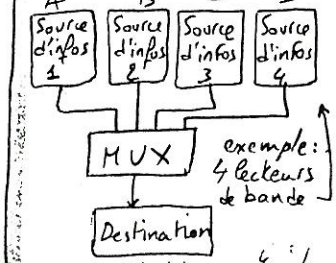
Sélection d'un mot binaire parmi plusieurs mots binaires

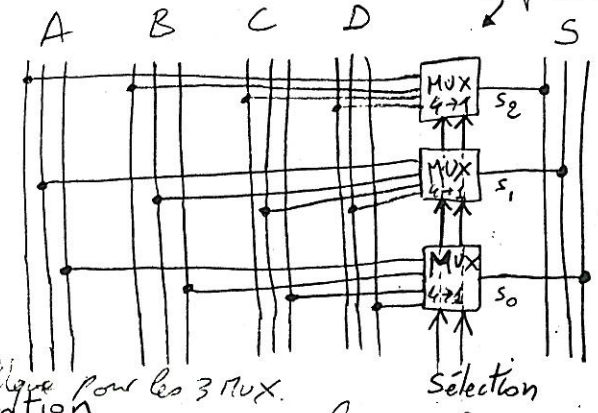
ex : sélection d'un mot de 3 bits parmi 4 mots de 3 bits :

⟹ il faut autant de multiplexeurs qu'il y a de bits dans le mot (ici 3 multiplexeurs) :

4 mots de données :

A, B, C, D





La sélection est identique pour les 3 MUX.

12.4.4.3.2 Génération

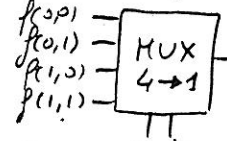
Matérialisation d'une fonction logique :

Toute fonction logique peut se mettre sous forme canonique

→ ex : à 2 variables : $f(a,b) = \bar{a}\bar{b}.f(0,0) + \bar{a}b f(0,1) + a\bar{b} f(1,0) + ab f(1,1)$

avec : $f(i,j)$ = valeur particulière de la fonction logique f lorsque $a = i$ et $b = j$

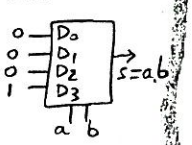
→ utilisation d'un multiplexeur à 4 entrées de données, donc 2 entrées d'adresse :



ab =
$\bar{a}\bar{b}.0 + \bar{a}b.0$
$+ a\bar{b}.0 + ab.1$

ex : opérateur ET :

a	b	S = ab
0	0	D0 = 0
0	1	D1 = 0
1	0	D2 = 0
1	1	D3 = 1



→ Le multiplexeur est un opérateur programmable.

Pour une fonction de n variables, il faut un multiplexeur à 2^n entrées de données donc n entrées d'adresse : MUX $2^n \to 1$

(10)

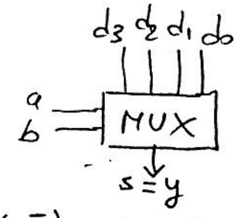
On ne fait pas les regroupements pour pouvoir utiliser un multiplexeur :

On a :

$y = ab\bar{c}\bar{d} + a\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}cd + \bar{a}bcd + \bar{a}\bar{b}c\bar{d} + \bar{a}bc\bar{d}$

On choisit par exemple a b en entrées d'adresse :

a b : a noté a_1 (MSB), b noté a_0 (LSB)



$y = \bar{a}\bar{b}.(cd + c\bar{d}) + \bar{a}.b.(cd + c\bar{d}) + a\bar{b}(\bar{c}.\bar{d}) + ab(\bar{c}.\bar{d})$

$= \sum d_i . m_i$ $\quad y = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}\bar{c}\bar{d} + ab\bar{c}\bar{d}$

Sur d_0 : on doit appliquer : $cd + c\bar{d} = c$

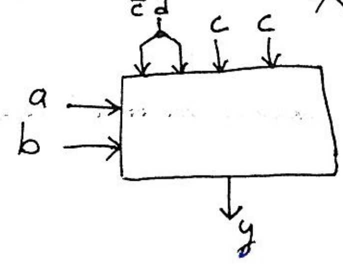
— d_1 : ——— : $cd + c\bar{d} = c$

— d_2 : ——— : $\bar{c}\bar{d}$

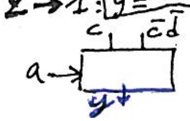
— d_3 ——— : $\bar{c}\bar{d}$

avantage : réduction du nombre de variables :

on est passé d'un problème à 4 variables abcd à un problème à 2 variables ~~ab~~. cd



Vu que b se simplifie, on pourrait même utiliser un MUX 2 → 1 : $y = c\bar{a} + \bar{c}\bar{d}a$



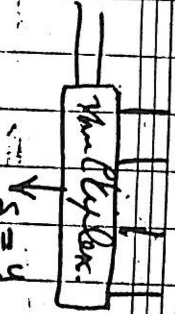
On pourrait toujours faire comme dans l'exercice précédent (6) utiliser 4 variables pour les adresses du MUX (le MUX est alors un MUX 16→1) avec en entrées de données des cst des 1, mais pour réduire la taille du MUX, on peut entrer des données variables

Soit un pb. donné : voici sa table de Karnaugh →

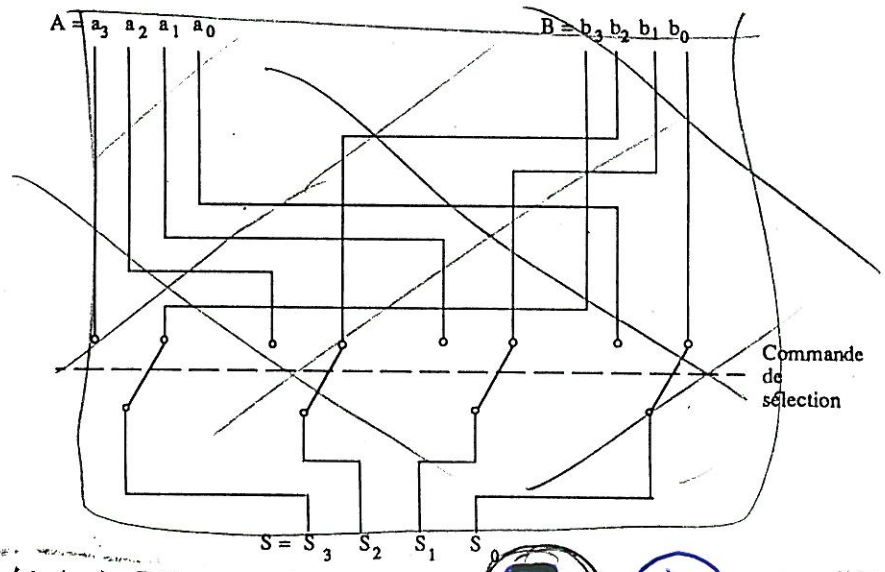
(y)

cd \ ab				
	0	0	1	1
	0	0	0	0
	1	1	0	0
	1	1	0	0

On peut utiliser un multiplexeur qui donne en sortie (y) voulue.



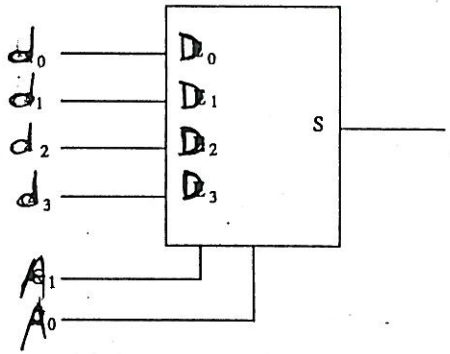
$y = c\bar{a} + \bar{c}\bar{d}a$



Conversion parallèle-série

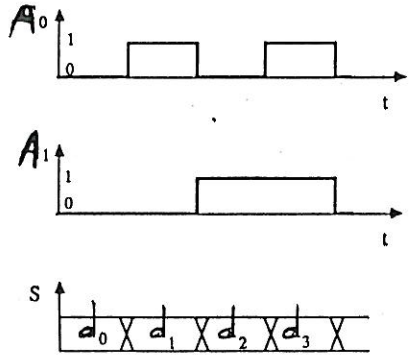
Soit un mot binaire $D = d_3 d_2 d_1 d_0$ disponible en mode parallèle, c'est-à-dire sur quatre fils, chaque fil étant affecté à un élément binaire du mot.

Pour transmettre les éléments binaires en série, c'est-à-dire les uns à la suite des autres sur un seul fil, il faut d'abord transmettre d_0, puis d_1 puis d_2 et enfin d_3. Ceci revient à sélectionner (ou à aiguiller) l'un des éléments binaires de D sur le fil unique de sortie série. Le multiplexeur est capable d'effectuer cette tâche si les combinaisons correspondantes sont placées successivement sur les commandes de sélection.



Dans le premier temps il faut que $A_1 = A_0 = 0$ pour que $S = D_0 = d_0$.
Ensuite A_0 passe à 1 ce qui impose $S = D_1 = d_1$.
Puis $A_1 = 1$ et $A_0 = 0$ d'où $S = D_2 = d_2$.
Et enfin $A_1 = A_0 = 1$ alors $S = D_3 = d_3$.

D'où le chronogramme :



Génération de fonctions

Toute fonction logique combinatoire peut se mettre sous forme canonique (*cf.* § 2.2.1, théorèmes d'expansion de Shannon). Par exemple, une fonction de deux variables A et B se développe suivant l'expression

$$f(A,B) = \overline{A}\,\overline{B} f(0,0) + \overline{A} B f(0,1) + A \overline{B} f(1,0) + A B f(1,1)$$

où $f(i,j)$ est la valeur particulière de la fonction logique lorsque $A = i$ et $B = j$.

Un multiplexeur à quatre entrées, donc deux commandes, délivre une sortie S reliée aux commandes C_1 C_0 par la relation

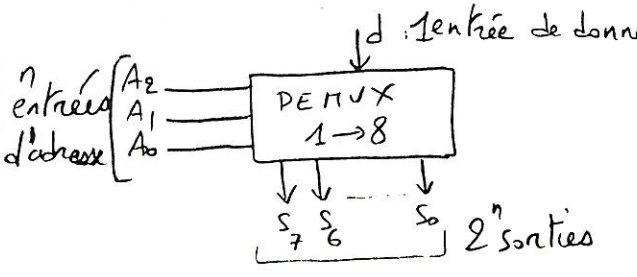
$$S = \overline{C_1}\,\overline{C_0} E_0 + \overline{C_1} C_0 E_1 + C_1 \overline{C_0} E_2 + C_1 C_0 E_3$$

En comparant les deux relations précédentes, il est facile de voir qu'il est possible de réaliser toutes les fonctions de deux variables en identifiant S et $f(A,B)$, C_1 et A, C_2 et B, les valeurs des entrées du multiplexeur et celles de la fonction.

Les entrées de sélection du multiplexeur sont alors les variables de la fonction, et les entrées du multiplexeur permettent de sélectionner la fonction à réaliser.

4.5 Le Démultiplexeur

12.4.5.1 Définition : aiguille 1 donnée sur 1 parmi 2^n sorties :



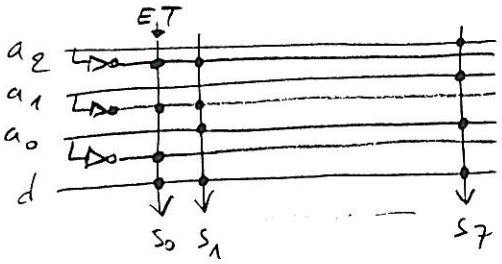
$A_2 A_1 A_0$	S_i
0 0 0	$S_0 = d$
0 0 1	$S_1 = d$
1 1 1	$S_7 = d$

Les sorties non aiguillées sont non activées par convention on les suppose au niveau 0 (en fait, ça peut être 0 ou 1 suivant la technique utilisée par le constructeur)

12.4.5.2 Equation des sorties : $S_i = m_i . d$ ← minterme

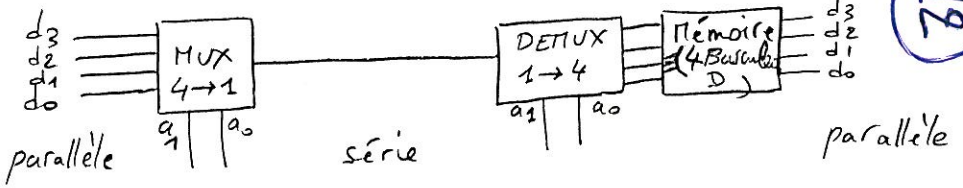
ex : $S_3 = \bar{a}_2 a_1 a_0 . d$

12.4.5.3 Réalisation : (Représentation Symbolique)



12.4.5.4 Applications

12.4.5.4.1 Conversion série / parallèle

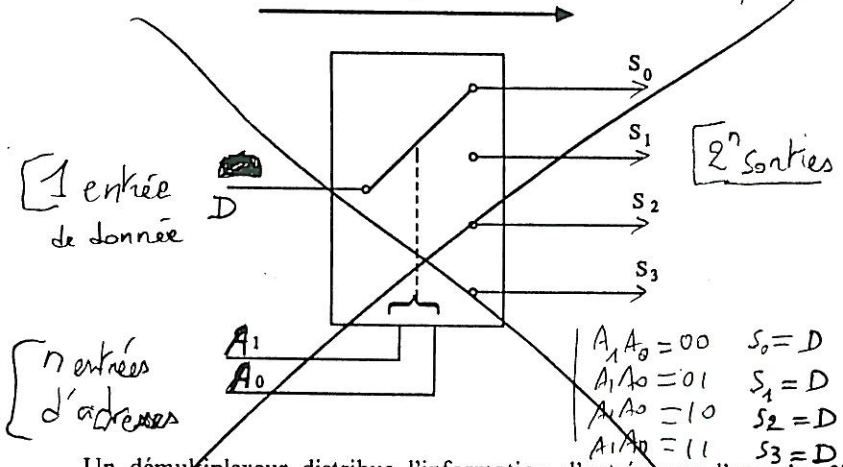


12.4.5.4.2 Affichage multiplexé

Soit 4 chiffres à afficher ; on peut afficher les chiffres l'un après l'autre très vite pour donner l'impression de simultanéité à l'oeil

12

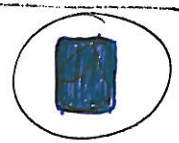
Le démultiplexeur aiguille 1 Donnée sur 1 parmi 2^n sorties



Un démultiplexeur distribue l'information d'entrée vers l'une des 2^n sorties, la sélection de la sortie concernée étant effectuée par n variables de commande. Les autres sorties sont alors dans un état de repos.

Lorsque l'entrée est toujours égale à 1, le démultiplexeur fonctionne comme un décodeur binaire.

12.4.6 3.10. LES CIRCUITS INTÉGRÉS ARITHMÉTIQUES



13

12.4.6.1 3.10.1. L'ADDITIONNEUR

C'est le circuit réalisant l'addition de deux nombres binaires. La table d'addition de deux nombres à un élément binaire est la suivante :

a \ b	0	1
0	0	1
1	1	10

r Σ

Le résultat de l'opération comporte deux parties :

— la somme Σ :

a \ b	0	1
0	0	1
1	1	0

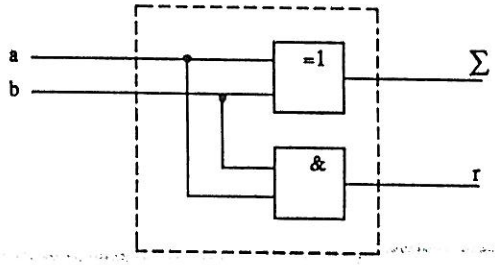
$\Sigma = a \oplus b$

— et la retenue r :

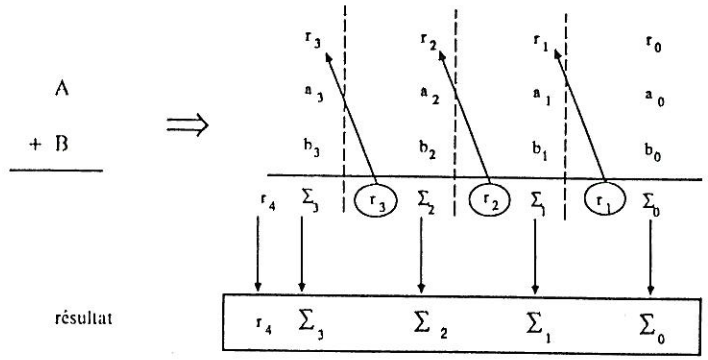
a \ b	0	1
0	0	0
1	0	1

r = ab

Le circuit élémentaire réalisant cette opération est le **demi-additionneur :**



Pour effectuer une addition de deux nombres binaires à plusieurs chiffres, une première technique consiste à additionner successivement les chiffres de même poids avec éventuellement la retenue de l'addition précédente.

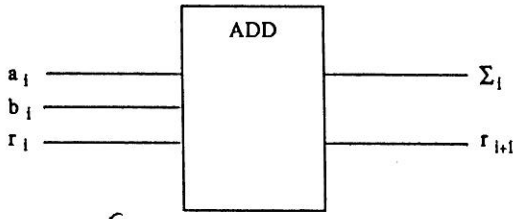


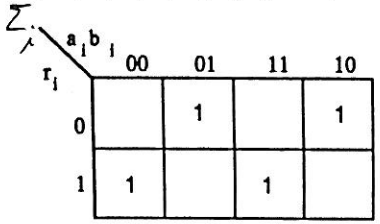
La structure de l'additionneur de deux nombres est alors répétitive. Une cellule élémentaire peut donc être utilisée pour chaque poids. Elle est appelée **additionneur complet.** L'addition globale est réalisée par la mise en cascade des cellules au sens des retenues.

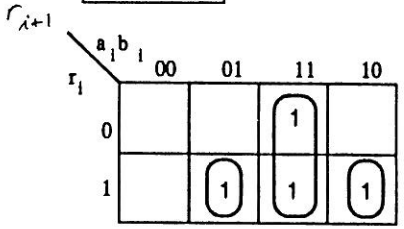
L'additionneur complet est défini par la table de vérité ci-après :

somme = $a_i + b_i + r_i$ (plus)

a_i	b_i	r_i	r_{i+1}	Σ_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



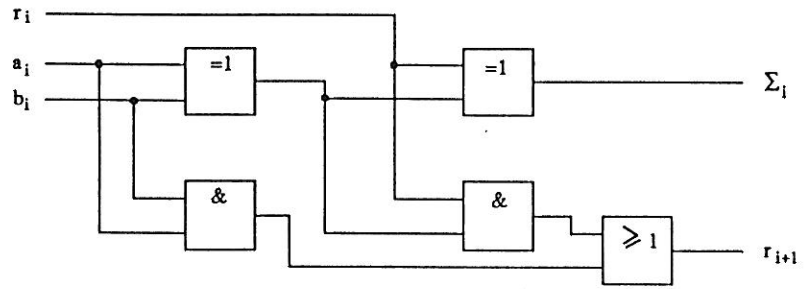




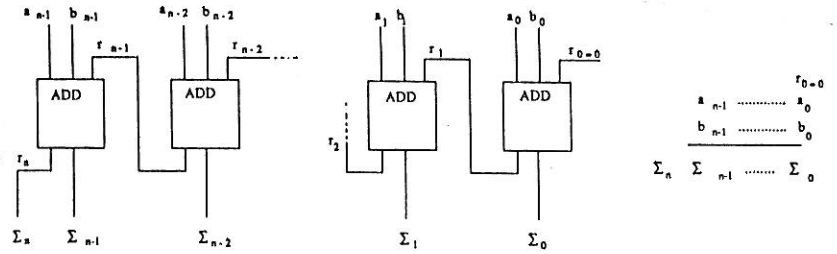
$\Sigma_i = a_i \oplus b_i \oplus r_i$

$r_{i+1} = a_i b_i + r_i (a_i \oplus b_i)$

Ce qui donne le schéma :



L'addition de deux nombres de n éléments binaires nécessite n additionneurs complets, la retenue appliquée sur les plus faibles poids est nulle et chaque retenue calculée est appliquée au chiffre de poids immédiatement supérieur.

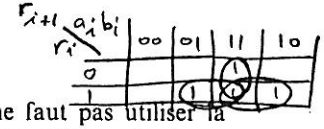


Cette solution est intéressante d'un point de vue du matériel parce que répétitive. Par contre, comme le résultat d'une addition ne peut pas être obtenu instantanément, le temps maximum mis pour obtenir le résultat est directement proportionnel au nombre d'additionneurs. En effet, après le premier temps de calcul la retenue r_1 est appliquée au second additionneur. Ce n'est qu'après le second temps de calcul que la retenue r_2 est délivrée et ainsi de suite, jusqu'au dernier additionneur. Pour cette raison, l'addition ainsi réalisée porte le nom d'« **additionneur à propagation de la retenue** » ou « **additionneur à retenue série** ».

Pour éliminer cet inconvénient, la seconde technique consiste à calculer toutes les retenues en parallèle, directement à partir des données sans même calculer les sommes partielles. Le circuit ainsi réalisé est alors appelé « **additionneur à retenue anticipée** ».

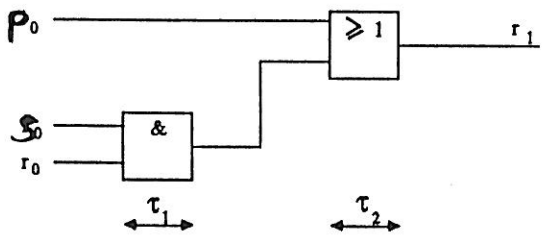
En reprenant le tableau de Karnaugh relatif au calcul de la retenue il vient :

$$r_{i+1} = a_i b_i + r_i(a_i + b_i)$$



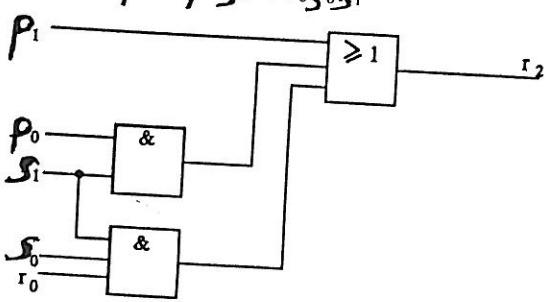
Afin d'éviter des temps de calcul cumulatifs, il ne faut pas utiliser la relation en tant que relation de récurrence, c'est-à-dire qu'il ne faut pas utiliser un résultat de calcul pour le calcul suivant. Il faut systématiquement recalculer chaque terme, ce qui donne en posant $\Sigma_i = a_i + b_i$ et $P_i = a_i b_i$:

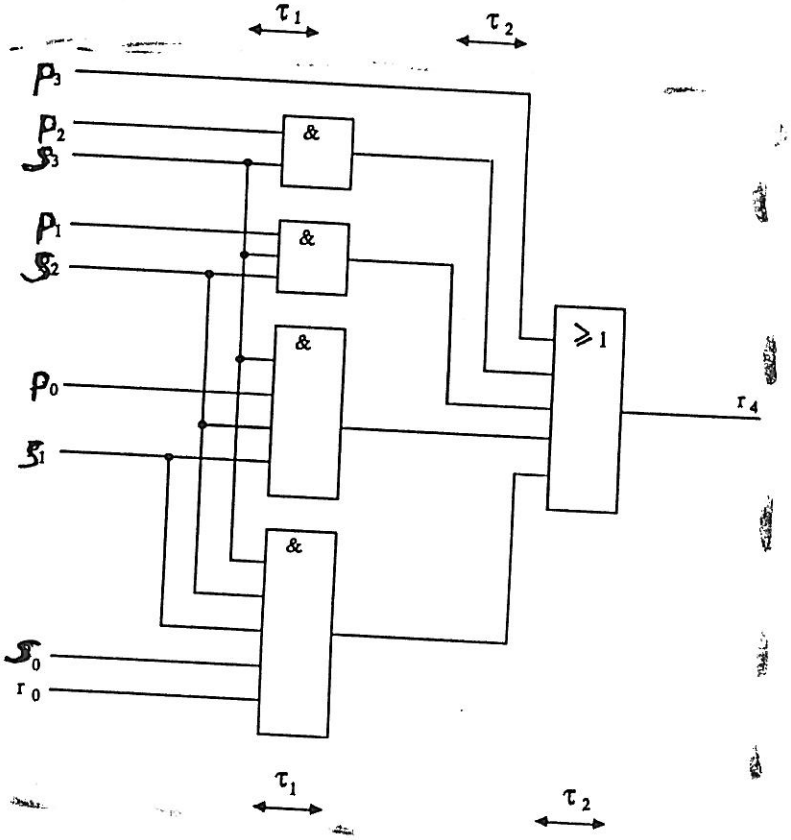
$$r_1 = P_0 + r_0 \Sigma_0$$



De même :

$$r_2 = p_1 + r_1 s_1 = p_1 + (p_0 + r_0 s_0) s_1$$
$$r_2 = p_1 + p_0 s_1 + r_0 s_0 s_1$$





Et ainsi de suite :

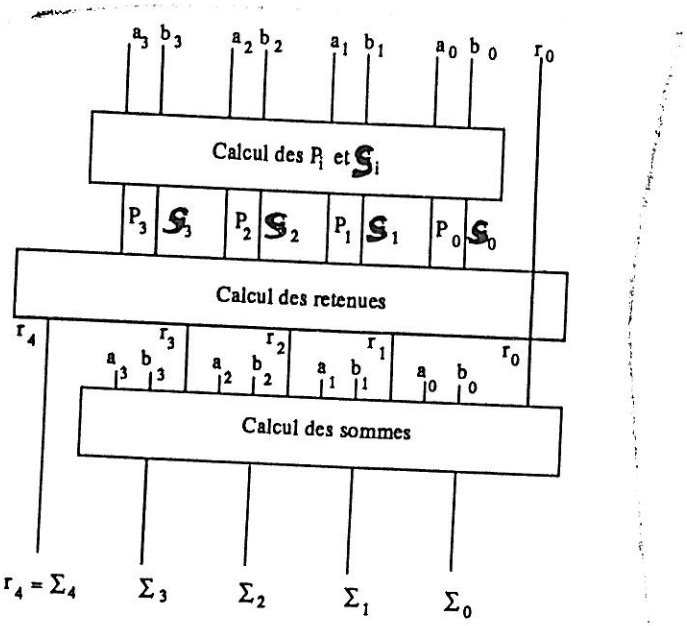
$$r_3 = p_2 + r_2 s_2 = p_2 + (p_1 + p_0 s_1 + r_0 s_0 s_1) s_2$$
$$= p_2 + p_1 s_2 + r_0 s_0 s_1 s_2$$

et :

$$r_4 = p_3 + r_3 s_3$$
$$= p_3 + p_2 s_3 + p_1 s_2 s_3 + p_0 s_1 s_2 s_3 + r_0 s_0 s_1 s_2 s_3$$

On constate que les temps de calcul des retenues sont tous égaux. Ils correspondent au temps de transit de l'information dans une porte ET (τ_1) et une porte OU (τ_2) en cascade (le nombre d'entrée d'une porte n'affectant pas son temps de transit).

La structure d'un additionneur quatre éléments binaires utilisant la technique de calcul anticipé des retenues est la suivante :



Afin d'illustrer le gain apporté par le principe de la retenue anticipée, le tableau ci-après donne les temps de calcul pour une addition de deux nombres de formats différents :

Format de chaque nombre en bits	Temps de calcul en nS (logique TTL série N)		
	Propagation de la retenue	Retenue anticipée	
4	24	24	avec un additionneur 4 bits intégrés
8	36	36	avec utilisation d'un générateur de retenue
12	48	36	
16	60	36	
64	192	60	avec deux générateurs de retenue en cascade

3. LE COMPARATEUR

Un comparateur est un dispositif capable de détecter l'égalité de deux nombres et éventuellement d'indiquer le nombre le plus grand ou le plus petit.

3.1. Principe

Pour effectuer la comparaison de deux nombres, deux techniques sont couramment utilisées :

— La soustraction des deux nombres. Si le résultat de l'opération $A - B$ est positif, cela signifie que A est supérieur à B. Si le résultat est nul, les deux nombres sont égaux.

— Une comparaison bit à bit. C'est cette méthode qui est utilisée dans la plupart des circuits intégrés commercialisés. La comparaison s'effectue poids à poids en commençant par le chiffre le plus significatif.

Les nombres A et B ayant le même format, le nombre A est forcément supérieur à B si son élément binaire le plus significatif (MSB) est supérieur à celui de B. Si ces deux éléments binaires sont égaux, la supériorité (ou l'infériorité) ne peut être déterminée que par l'examen des bits de poids immédiatement inférieur et ainsi de suite.

L'examen des poids successifs s'arrête dès que l'un des éléments binaires est supérieur ou inférieur à l'autre. Les deux nombres A et B sont égaux si, après avoir examiné tous les éléments binaires, il n'a pas été détecté de supériorité ou d'infériorité.

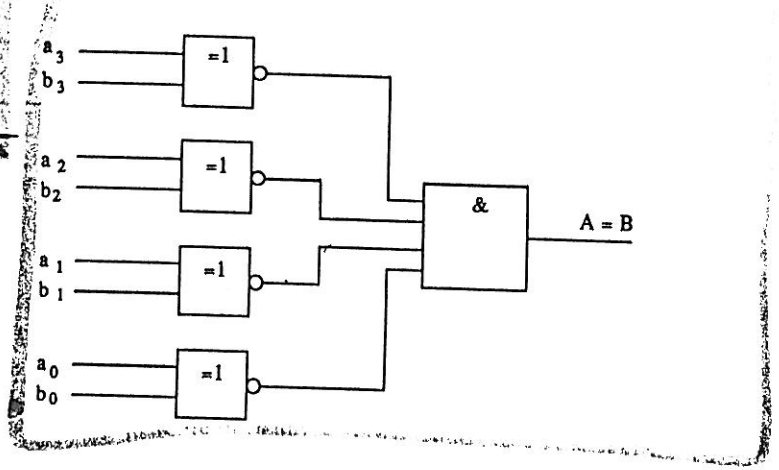
3.1.1. Comparateur donnant l'égalité de deux nombres

C'est le comparateur le plus simple. Deux nombres sont égaux si tous les chiffres sont égaux deux à deux. Pour détecter l'égalité de deux éléments binaires, un opérateur OU exclusif complémenté est indispensable. Un opérateur ET indique la simultanéité de toutes les inégalités partielles.

Soient deux nombres A et B de quatre éléments binaires chacun, $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$:

$A = B$ si $(A_3 = b_3)$ ET $(a_2 = b_2)$ ET $(a_1 = b_1)$ ET $(a_0 = b_0)$

Ce qui donne le schéma :



3.2. Comparateur complet

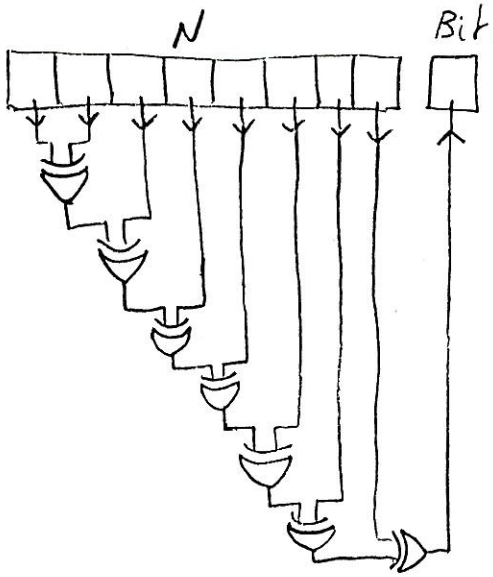
Par analogie avec l'additionneur, la conception d'un comparateur complet pour des nombres de quatre éléments binaires peut se faire de deux façons différentes.

— **Première solution :** En cascade, c'est-à-dire avec propagation des égalités partielles. Les poids de A et de B sont comparés en commençant par le plus élevé. La comparaison sur les poids faibles ne peut être faite que si tous les bits de poids plus élevés sont égaux deux à deux.

12.4 6.4. Générateur de Parité

On appelle parité d'un mot binaire N le nombre de 1 contenus dans le mot le mot a une parité paire si ce nombre de 1 est pair. Afin de rendre les transmissions numériques plus robustes au bruit, on adjoint un bit à tous les mots transmis. Ce bit, dit de parité, est choisi de façon à ce que le mot complet formé du mot et du bit de parité soit pair.

Le principe utilisé pour générer ce bit de parité repose sur la propriété du OU exclusif : $a \oplus b \oplus c \oplus \dots \oplus m$ vaut 1 si un nombre impair de variables est au niveau 1 :



rappel : $\overline{a \oplus b} = a\bar{b} + \bar{a}b$

$\overline{a \oplus b} = \bar{a}\bar{b} + ab = a \odot b$

k_1 : $m_2 m_1$ \ m_0	0	1
0 0	1	0
0 1	0	1
1 1	1	0
1 0	0	1

$k_1 = \bar{m}_2 \bar{m}_1 \bar{m}_0 + \bar{m}_2 m_1 m_0 + m_2 m_1 \bar{m}_0 + m_2 \bar{m}_1 m_0$

$= \bar{m}_2 (\bar{m}_1 \bar{m}_0 + m_1 m_0) + m_2 (m_1 \bar{m}_0 + \bar{m}_1 m_0)$

$= \bar{m}_2 (\overline{m_0 \oplus m_1}) + m_2 (m_0 \oplus m_1)$

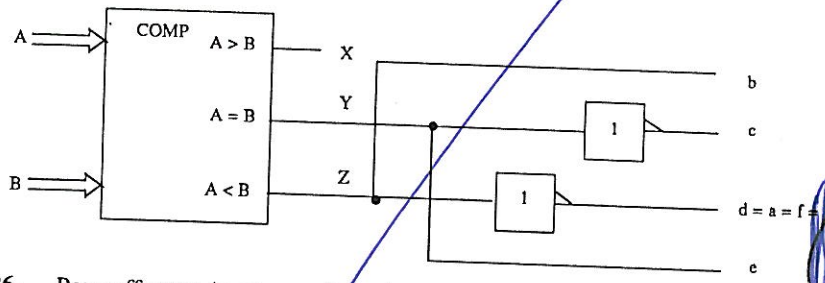
$$\boxed{k_1 = \overline{m_0 \oplus m_1 \oplus m_2}}$$ ($\oplus$ est associatif)

25. De la même façon que pour l'exercice 24, on obtient :

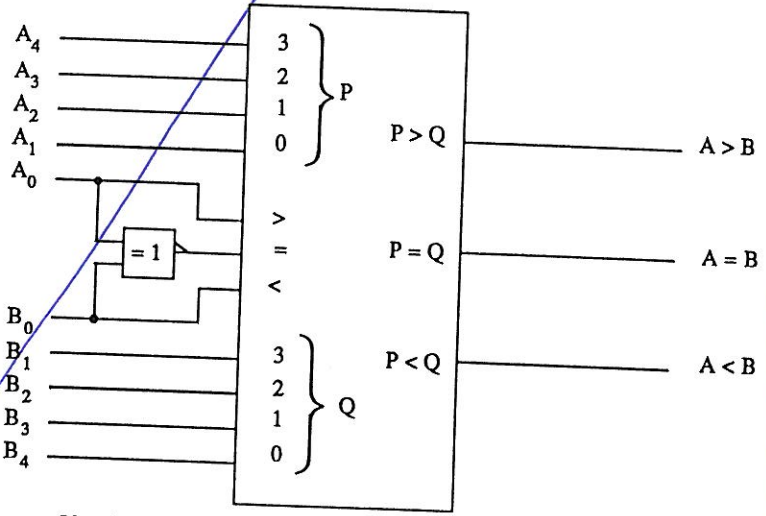
	X $(A > B)$	Y $(A = B)$	Z $(A < B)$	a	b	c	d	e	f	g
S	1	0	0	1	0	1	1	0	1	1
E	0	1	0	1	0	0	1	1	1	1
I	0	0	1	0	1	1	0	0	0	0

d'où les équations logiques :

$$a = X + Y = \overline{Z} = d = f = g$$
$$b = \underline{Z}$$
$$c = \overline{Y}$$
$$e = Y$$



26. Pour effectuer la comparaison de deux nombres de 5 bits à l'aide d'un seul comparateur 4 bits cascadable, il suffit d'utiliser ses entrées de mise en cascade pour l'une des variables. Compte tenu du schéma du 7485, c'est la variable de plus faible poids qui doit y être connecté.



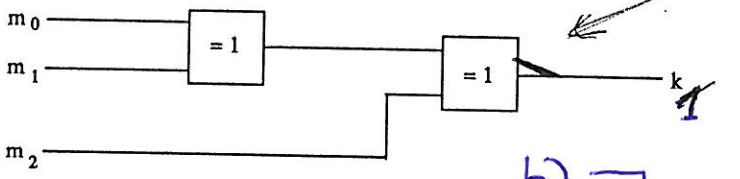
Un circuit de détection d'égalité doit néanmoins être rajouté pour valider l'égalité globale.

Sécuriser une communication se fait en ajoutant de la redondance au message.

14

m_2	m_1	m_0	k_1
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

a) $k_1 = \overline{m_0 \oplus m_1 \oplus m_2}$



b)

15

1°) k_1 est le seul élément binaire de contrôle entrant dans le test de parité T_1, donc k_1 doit être calculé à l'émission de tel façon que le nombre de 1 dans les éléments binaires 1, 3, 5, 7 soit pair.

m_1	m_2	m_4	k_1
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	∅
1	1	1	∅

m_1 \ $m_2 m_4$	00	01	11	10
0		1	∅	1
1	1		∅=1	

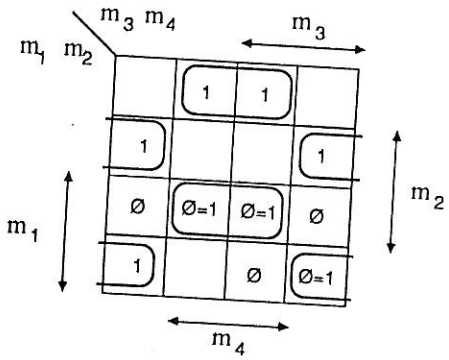
$$k_1 = m_1\overline{m}_2\overline{m}_4 + m_1m_2m_4 + \overline{m}_1\overline{m}_2m_4 + \overline{m}_1m_2\overline{m}_4$$
$$= m_1(\overline{m}_2\overline{m}_4 + m_2m_4) + \overline{m}_1(\overline{m}_2m_4 + m_2\overline{m}_4)$$
$$= m_1(\overline{m_2 \oplus m_4}) + \overline{m}_1(m_2 \oplus m_4)$$
$$= m_1 \oplus m_2 \oplus m_4$$

b) Convention :
T = 1 si erreur
T = 0 sinon

T \ $m'_0 k_1$ / $m'_2 m'_1$	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

$$T = \overline{m'_2 \oplus m'_1 \oplus m'_0 \oplus k_1}$$

Autre solution k_1 dépend des quatre éléments binaires du message. Le tableau de Karnaugh correspondant est rempli à partir du code de Hamming.



$$k_1 = m_1\overline{m}_2\overline{m}_4 + m_1m_2m_4 + \overline{m}_1m_2\overline{m}_4 + \overline{m}_1\overline{m}_2m_4$$
$$k_1 = m_1 \oplus m_2 \oplus m_4$$

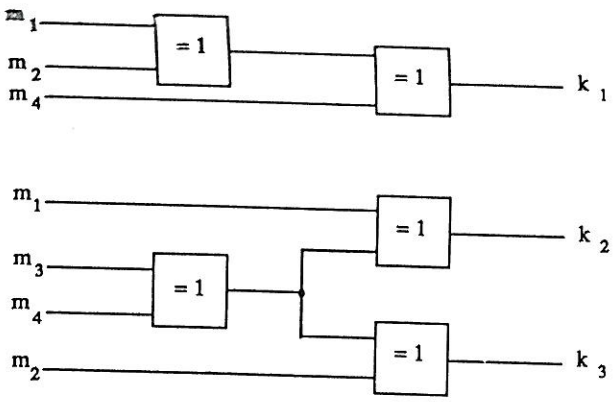
k_2 résulte du test de parité sur les éléments binaires 3. 6. 7. d'où :

$$k_2 = m_1 \oplus m_3 \oplus m_4$$

De même

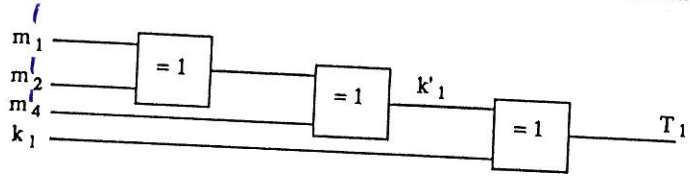
$$k_3 = m_2 \oplus m_3 \oplus m_4$$

Le schéma



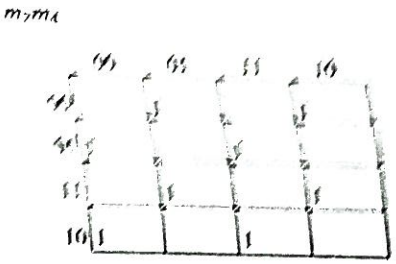
2°) Il y a deux solutions :

a) Fabriquer k'_1 à partir de m_1, m_2 et m_4 et comparer avec k_1 reçu de façon que $T_1 = 0$ si $k_1 = k'_1$ (et $T_1 = 1$ si $k_1 = k'_1$); d'où le schéma :



b) Calculer T_1 à partir des différentes possibilités de $m_1m_2m_4k_1$ (*eb* 1, 3, 5 et 7).

K_1	m_1	m_2	m_4	T_1	
0	0	0	0	0	nombre de 1 pair
0	0	0	1	1	nombre de 1 impair
0	0	1	0	1	
0	0	1	1	0	
0	1	0	0	1	
0	1	0	1	0	
0	1	1	0	0	
1	1	1	1	0	



Ici toutes les combinaisons sont prévues. Certaines sont impossibles s'il n'y a qu'une seule erreur.

Les 1 placés en diagonale indiquent une solution comportant des OU exclusifs.

$$T_1 = k_1 \oplus m_1 \oplus m_2 \oplus m_4$$

De même

$$T_2 = m_1 \oplus m_3 \oplus m_4 \oplus k_2$$

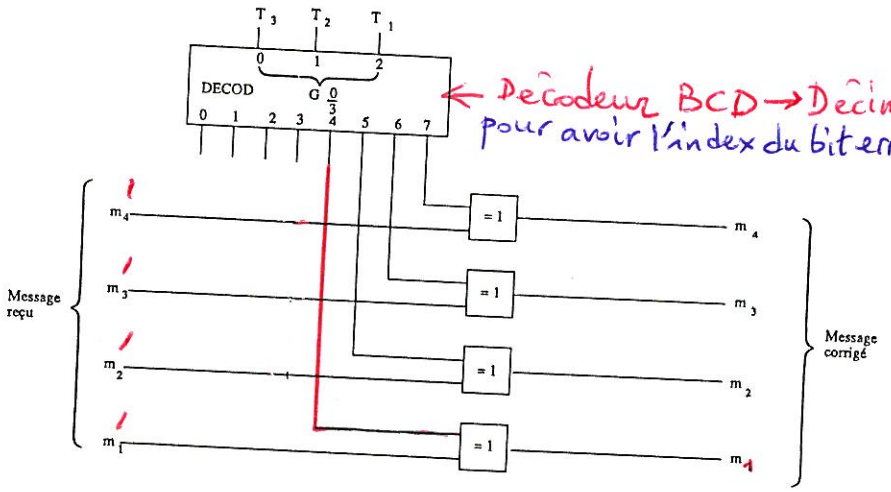
et

$$T_3 = m_2 \oplus m_3 \oplus m_4 \oplus k_3$$

Le schéma est identique à celui de l'émetteur auquel on ajoute trois OU exclusifs.

3°) $(T_3T_2T_1)_2$ indique le numéro de l'élément binaire erroné, donc celui qu'il faut corriger. La correction consiste simplement à changer l'élément binaire faux à l'aide d'un circuit complément.

Le OU exclusif permet d'être soit un circuit transparent soit un circuit inverseur suivant sa commande.



← Décodeur BCD → Décimal pour avoir l'index du bit erroné

Il n'est pas utile de corriger les éléments de contrôle.

Rappel : Pour inverser un bit b selon une commande c (c = 0 = pas d'inversion de b ; c = 1 = inversion du bit b), on utilise un circuit OU exclusif :

$s = b \oplus c = b\bar{c} + \bar{b}c$

| $c = 0 : s = b$
| $c = 1 : s = \bar{b}$

15

Bibliographie

Boole G., *The Mathematical Analysis of Logic*, Cambridge (réédité par Blackwell, 1948).

Boole G., *An Investigation of the Laws of Thought*, Londres (réédité par Dover Publications, 1954).

Karnaugh M., « The Map Method of Synthesis of Combinational Logic Circuits », *Communications and Electronics*, n° 9, November 1953, p. 593-599.

McCluskey E.J. Jr., « Minimization of Boolean Functions », *Bell System Technical Journal*, vol. 35, November 1956, p. 1417-1444.

McCluskey E.J. Jr., « Transients in Combinational Logic Circuits », *Redundancy Techniques for Computing Systems*, Spartan Books Co, 1962, p. 9-46.

McCluskey E.J. Jr., *Introduction to the Theory of Switching*, McGraw-Hill Book Co, New York, 1965.

McCluskey E.J. Jr. and Bartee T.C., *A Survey of Switching Circuits Theory*, McGraw-Hill Book Co, New York, 1962.

McCluskey E.J. Jr. and Schorr H., « Essential Multiple Output Prime Implicants », *Proceeding of the Symposium on the Mathematical Theory of Automata*, 1962, Polytechnic Press of the Polytechnic Institute of Brooklyn, New York, p. 437-457.

Quine W.V., « The Problem of Simplifying Truth Functions », *Am. Math. Monthly*, vol. 59, October 1952, p. 521-531.

Quine W.V., « A Way to Simplify Truth Functions », *Am. Math. Monthly*, vol. 62, November 1955, p. 627-631.

Quine W.V., « On Cores and Prime Implicants of Truth Functions », *Am. Math. Monthly*, vol. 66, November 1959, p. 755-760.

Shannon C.E., « A Symbolic Analysis of Relay and Switching Circuits », *Transactions of the AIEE*, vol. 57, 1938, p. 713-723.

Shannon C.E., « The Synthesis of Two-Terminal Switching Circuits », *Bell System Technical Journal*, vol. 28, 1949, p. 59-98.

Tison P., « Recherche de termes premiers d'une fonction booléenne », *Automatisme*, tome IX, n° 1, janvier 1964.

Tison P., *Théorie des consensus et algorithmes de recherche des bases premières*, Colloque d'algèbre de Boole, Grenoble, janvier 1965.

Tison P., « Algèbre booléenne : théorie des consensus. Recherche des bases premières d'une fonction booléenne », *Automatisme*, tome X, n° 6, juin 1965.

Veith E.W., « A Chart Method for Simplifying Truth Functions », *Proc. ACM*, May 2-3, 1952, p. 127-133.

Documents construits