ITI 1500 Hiver 2013

Systèmes Numériques I

Cours

Lundi	11:30 - 13:00	SCS E217
Jeudi	13:00 - 14:30	SCS E217

TUTORIAL Mardi 8 :30-10:00 salle: LMX-221

Professeur : Dr A. Karmouch, Bureau: CBY-A508

Chapitre 4

Circuits Logiques Combinatoires

circuits logiques Combinatoires -Définitions

→ Fournissent en sortie les fonctions logiques des entrées
→Les nouvelles sorties apparaissent immédiatement après
Un délai de propagation
→ pas de boucle de retour
→ pas d' horloge

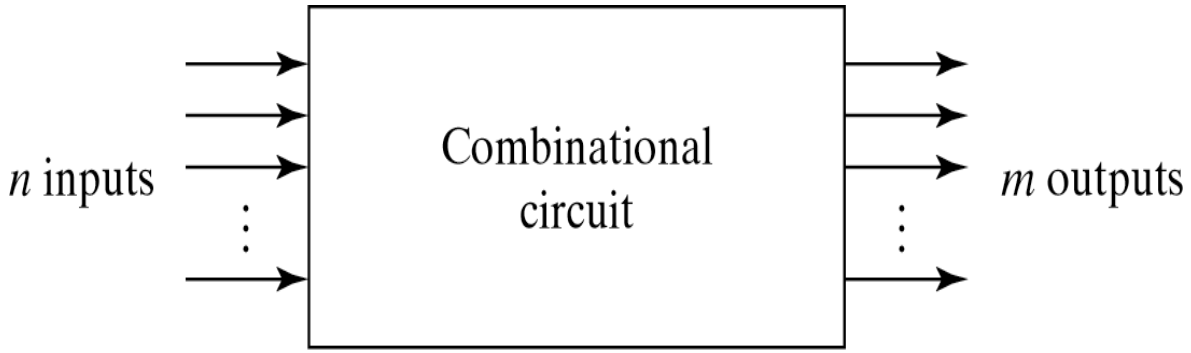


Fig. 4-1 Block Diagram of Combinational Circuit

Procédure de conception

Conception de circuits a partir de spécification.

1. Déterminer le nombre d'entrées et de sorties
2. Obtenir la table de vérité
3. Obtenir la fonction simplifiée
4. Dessiner le circuit et vérifier que son fonctionnement est correcte

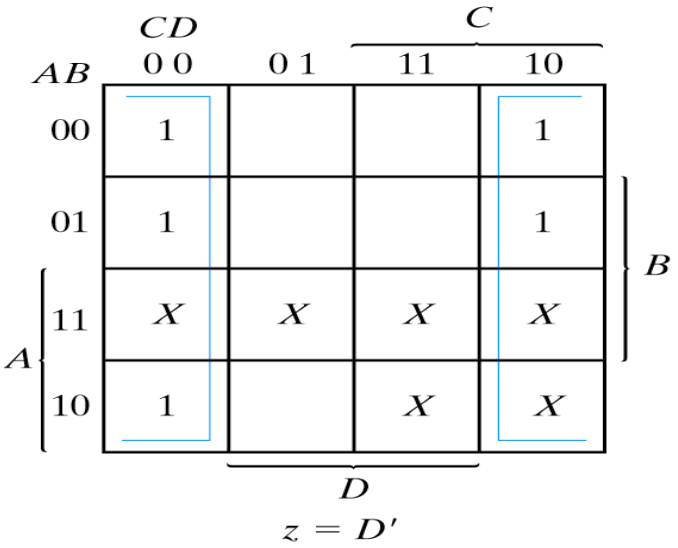
Circuit pour conversion de Code majoré de 3

- Ce circuit traduit un code binaire vers un autre code binaire

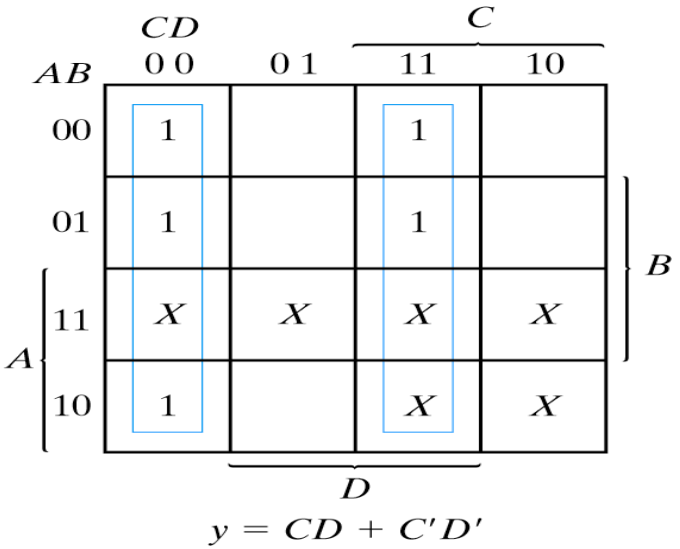
- Exemple 3-2: DCB vers code majoré 3
  - Code majoré 3 : décimal code + 3
  - DCB entrées de 1010 a 1111 sont indéterminés

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

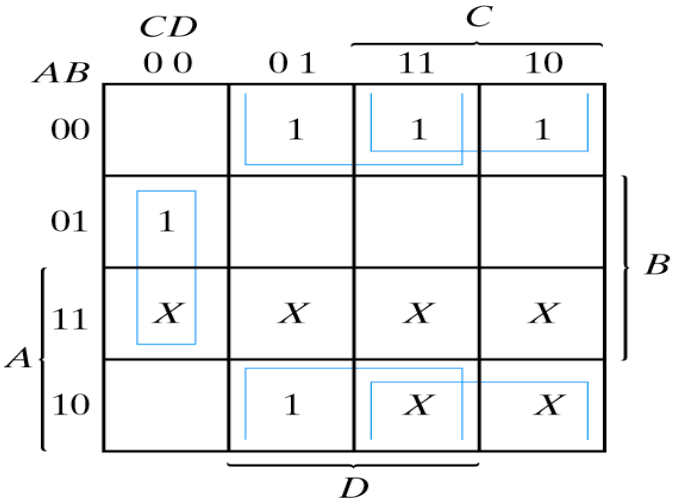
Simplification avec Diagramme de Karnaugh



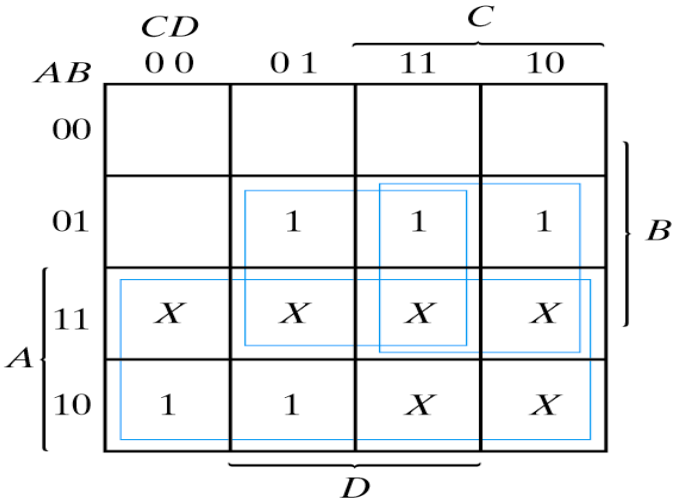
$z = D'$



$y = CD + C'D'$



$X = B'C + B'D + BC'D'$



$w = A + BC + BD$

Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Manipulation de la fonction simplifiée

- La réalisation a deux niveaux peut être obtenue directement a partir de la fonction simplifiées par Karnaugh
- La fonction simplifiée peut être manipulée afin de mettre en évidence des portes communes qui peut être utilisées comme entrée des autres portes dans le circuit a plusieurs sorties.
- Réalisation a trois niveaux de portes

$$W = A + BC + BD = A + B(C + D)$$

$$X = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$

$$= B'(C+D) + B(C+D)'$$

$$Y = CD + C'D' = CD\,(C + D)'$$

$$Z = D'$$

Réalisation a trois niveaux

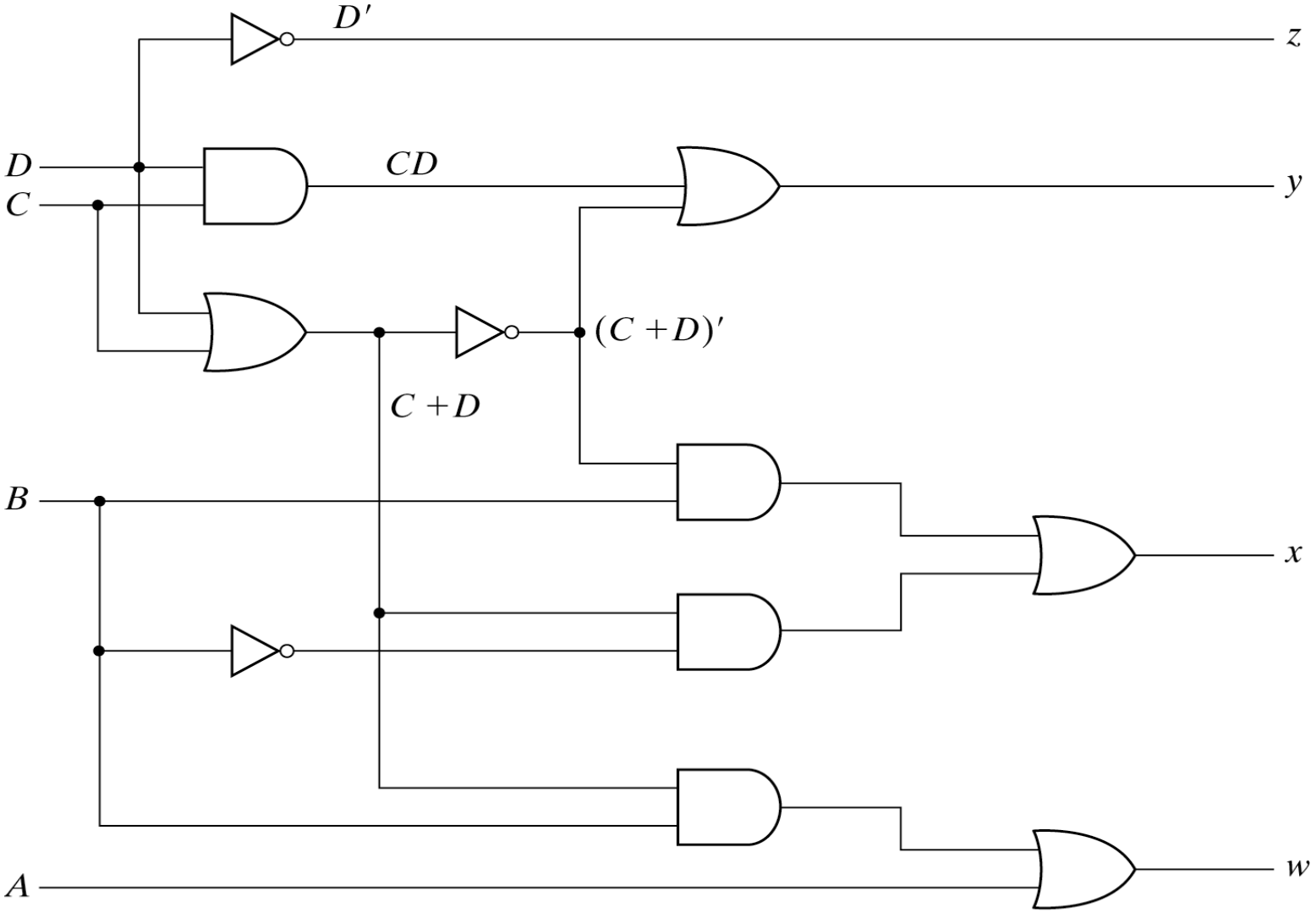


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

Circuits Logiques Combinatoires

-Additionneur & soustracteur Binaires

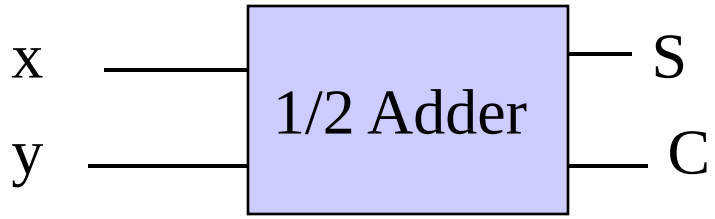
-Décodeur – Encodeur

- Multiplexeur - Demultiplexeur

→Le demi additionneur accepte deux chiffres binaires en entrée et produit deux chiffres binaires en sortie: le bit somme et le bit retenue.

Table de Vérité

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$\begin{array}{r} 1 \\ +\ 1 \\ \hline 1\ \ 0 \end{array}$$

C (retenue)

S(somme)

Demi-Additionneur (voir d'autres réalisations chapitre 2)

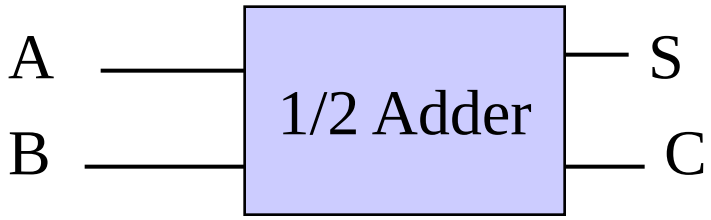
Table de vérité

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

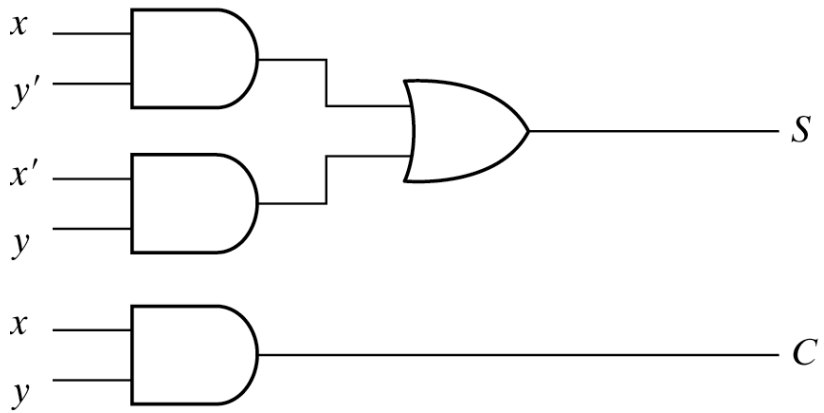
Somme de produits

$$S = x'y + xy'$$

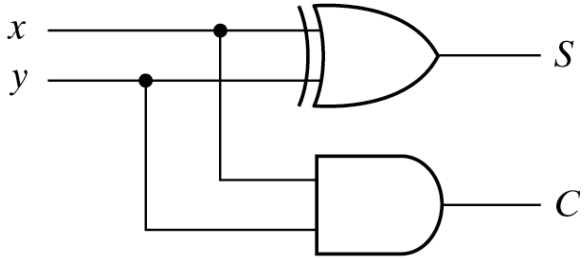
$$C = x.y$$



Demi-Additionneur (voir d'autres réalisations chapitre 2)



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

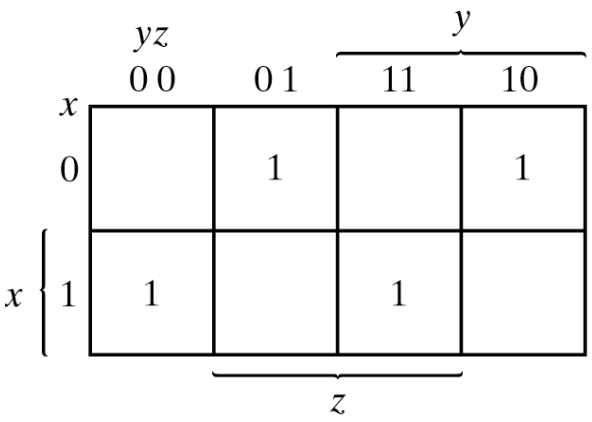
Réalisation du demi additionneur

Additionneur Complet

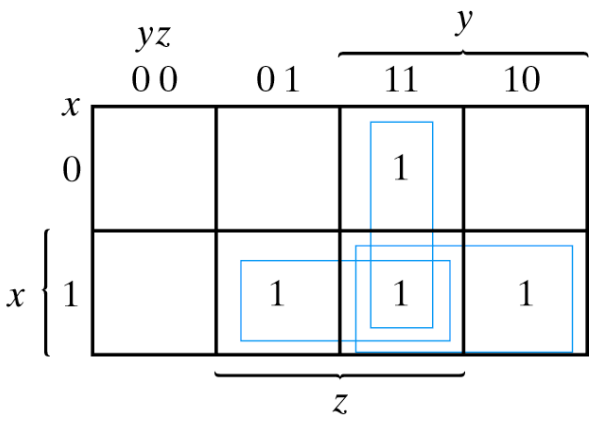
Table de vérité

x	y	z	C_o	S
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

Additionneur Complet- Diagramme de K



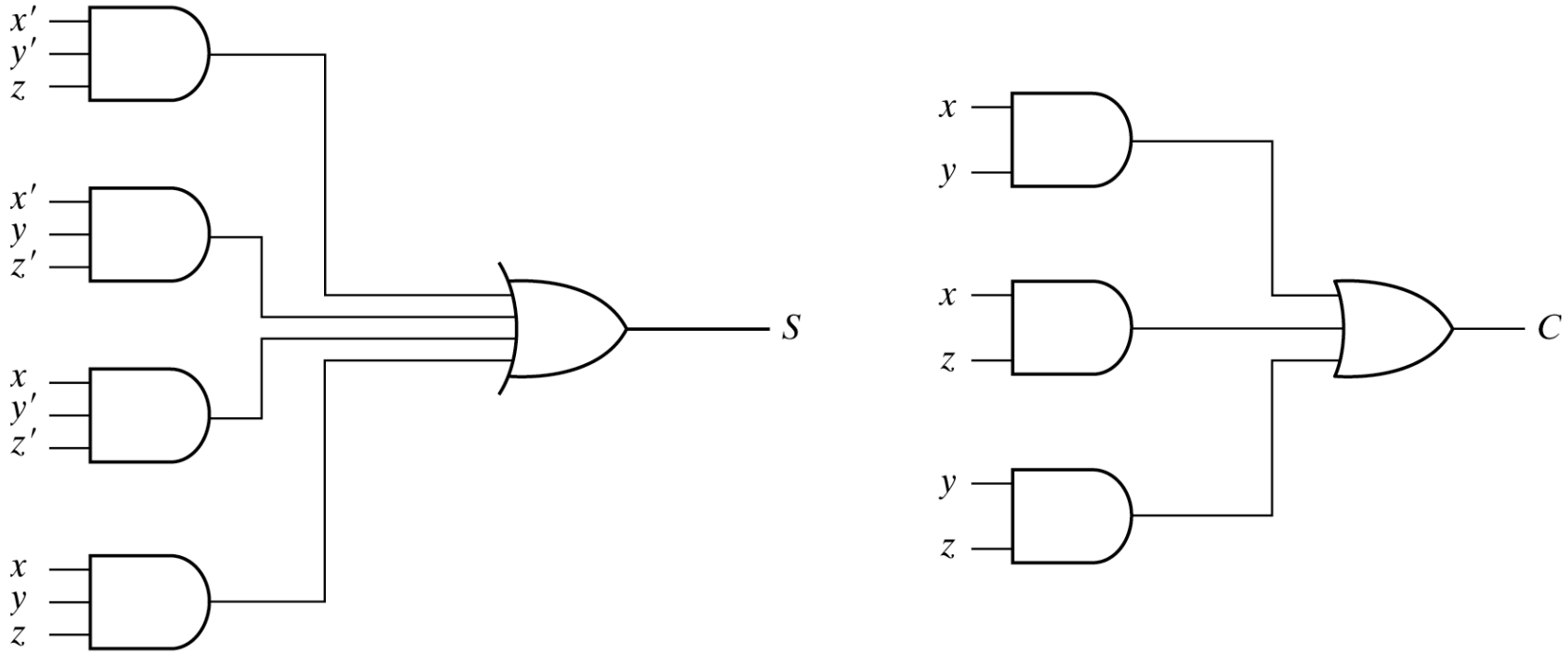
$S = x'y'z + x'yz' + xy'z' + xyz$



$S = xy + xz + yz$
$= xy + xy'z + x'yz$

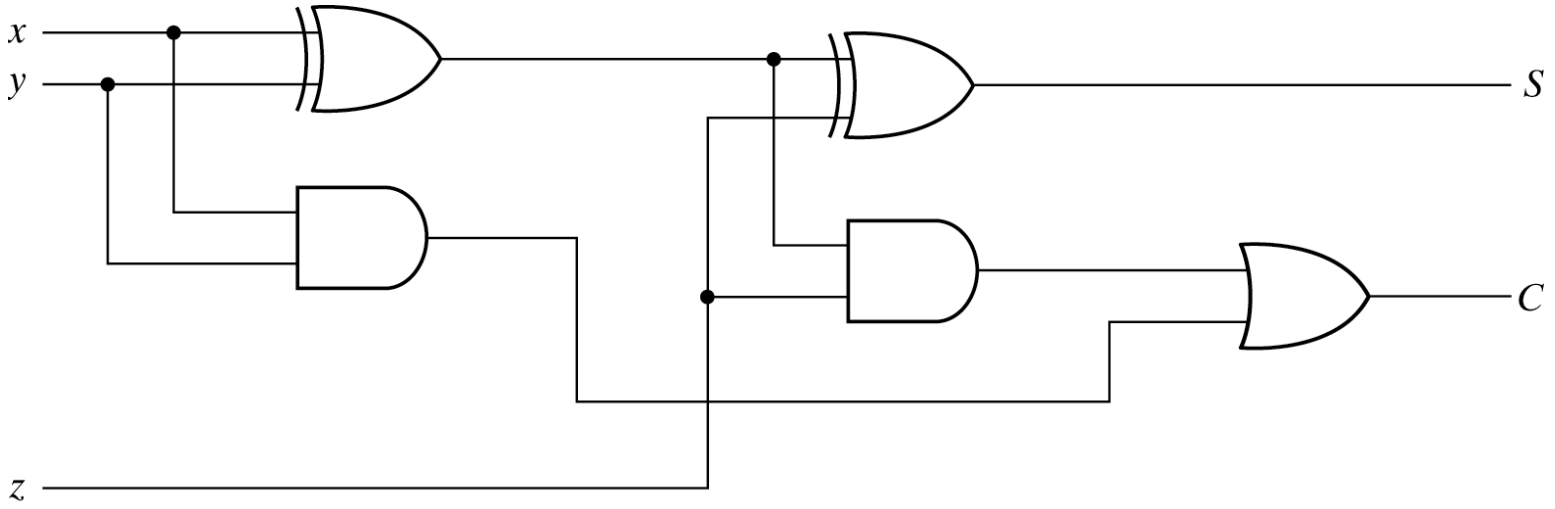
Diagramme pour l'additionneur complet

Additionneur Complet- réalisation: voir d'autres chapitre 2)



Réalisation somme de produits

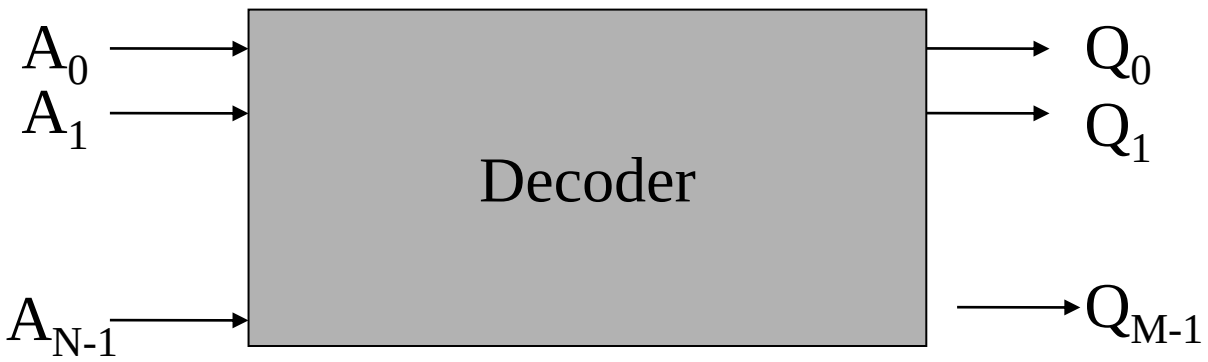
Additionneur Complet- réalisation: voir d'autres chapitre 2)



Réalisation avec deux demi- additionneur et une porte OU

Décodeur

- C'est un circuit logique qui accepte un ensemble d'entrées représentant un nombre binaire puis active uniquement la sortie qui correspond au nombre qui est en entrée.
- Si le nombre n-bit information codée possède des combinaisons non utilisées → moins de 2^n sorties.
  → *n-to-m* decodeur, $m \leq 2n$, Exemple: BCD-to-7-segments decoder, qui a $n=4$ et $m=7$

- 

Decodeur 2-vers-4

→Un décodeur 2-vers-4 fonctionne selon la table de vérité suivante:

- Les 2-bits en entrée sont appelés S1S0, et les 4 sorties sont Q0-Q3.
- Si l'entrée est i, alors la sortie Qi est vraie est unique

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- Par exemple: si S1 S0 = 10 (décimal 2), alors sortie Q2 est vrais, et Q0, Q1, Q3 sont toutes fausse.
- Ce circuit logique "décode" un nombre binaire en "un-des-quatre" code.

Construction d'un décodeur 2-vers-4

- On utilise la même procédure que pour construire un circuit logique combinatoire (comme vu précédemment). A partir de la table de vérité on obtenir une équation pour chacune des 4 sorties (Q0-Q3), en fonction des deux entrées (S0-S1).

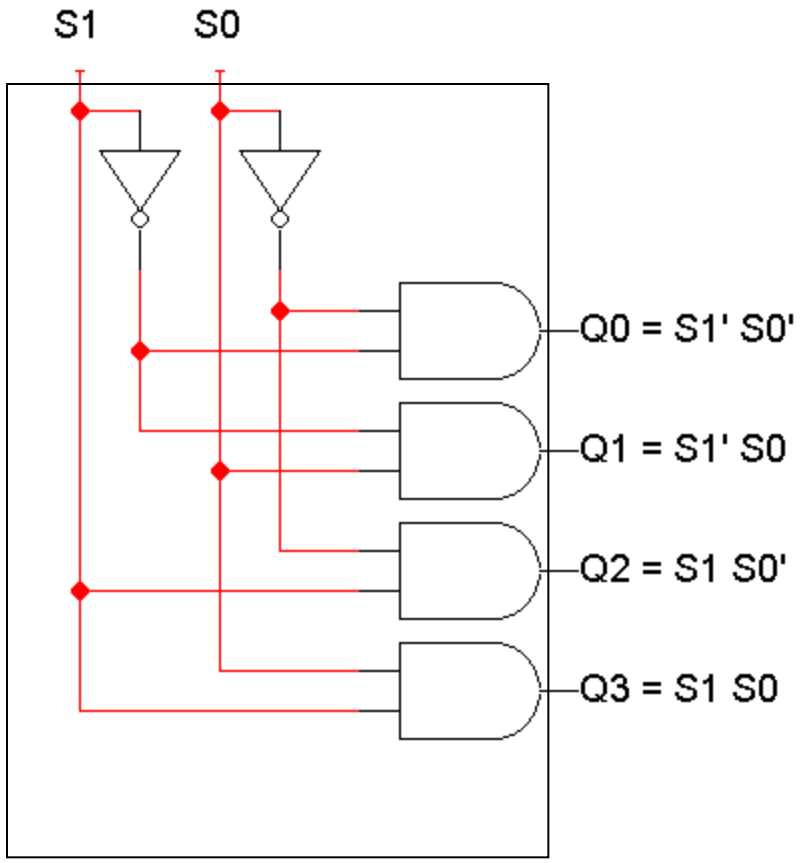
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- I l ya pas de simplification possible donc:

$$Q0 = S1' S0'$$
$$Q1 = S1' S0$$
$$Q2 = S1 S0'$$
$$Q3 = S1 S0$$

Implementation du decoder 2-vers-4

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Entrees Enable

- **Plusieurs dispositifs/composant ont une entrée additionnelle enable , qui est utilisée pour « activer » ou « désactiver » le composant/dispositif.**
- **Pour un décodeur,**
  - **EN=1 « active » le décodeur, pour un fonctionnement normal comme expliqué précédemment. Exactement une seule sortie sera a 1.**
  - **EN=0 "désactive" the décodeur. Par convention, cela indique que les sorties du décodeur sont a 0.**

→ On inclus cette entrée « EN »dans la table de vérité

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Table de vérité réduite

- Dans cette table on note que quand EN=0, les sorties sont toujours a 0, ***indépendamment des entrées*** S1 et S0.

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- On peut simplifier la table en représentant les entrées S1 et S0 par des X.

EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Décodeur- Générateur de Minterm

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Q0= S1′ S0′
Q1 = S1′ S0
Q2= S1 S0′
Q3= S1 S0

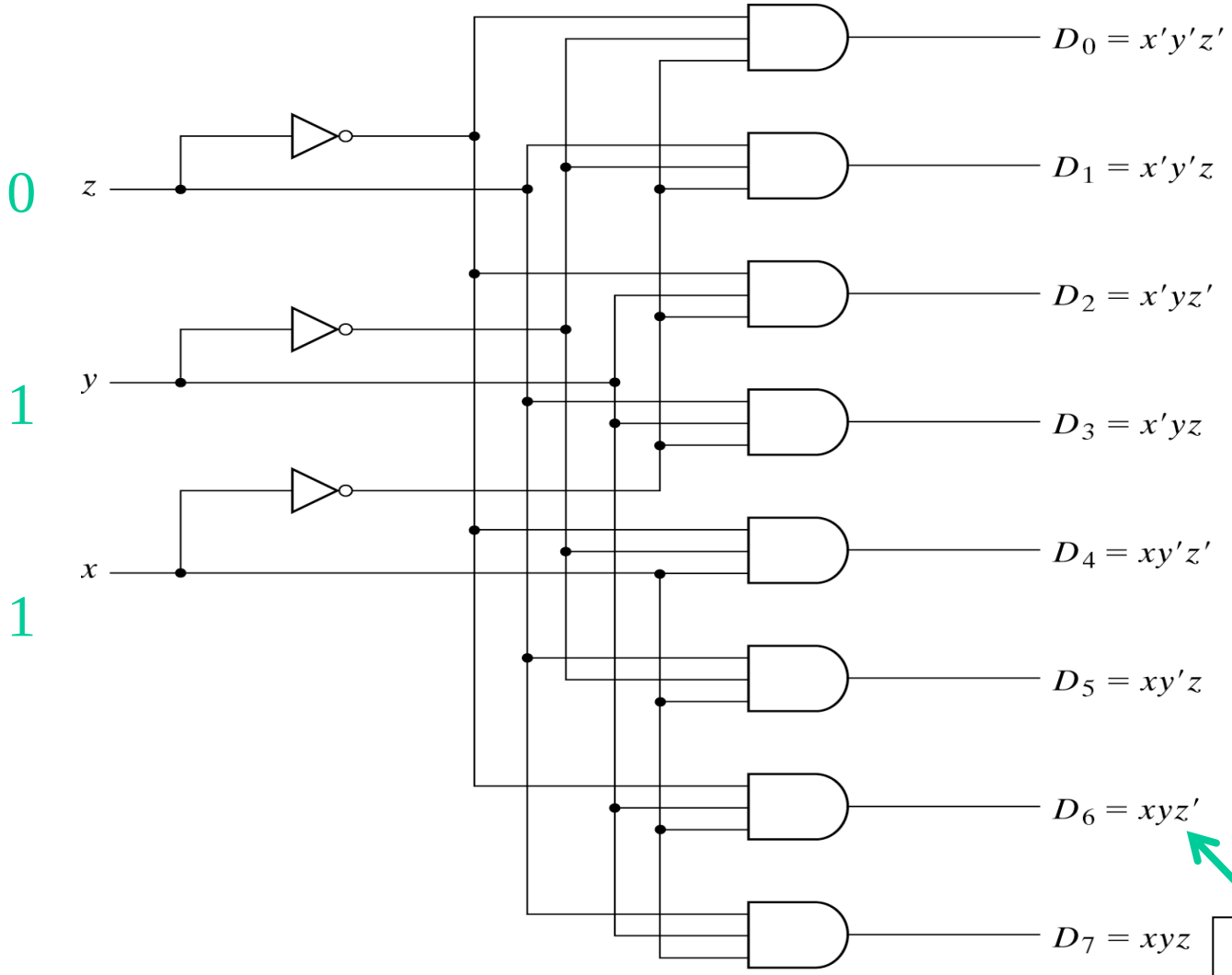
- Décodeurs sont appelés des générateurs de minterms.
  - Pour chaque combinaison d'entrée exactement une sortie est vraie.
  - Chaque équation de sortie contient toutes les variable d'entrées.
  - Ces propriétés sont valides pour les tailles de décodeurs.
- Donc on peut utiliser un décodeur pour implémenter des fonctions logiques arbitraires

  → Pour une somme de produit on peut utiliser les sorties Minterms du décodeur pour implémenter la fonction.

3- vers- 8 Decodeur

- Trois entrées, x, y, z, sont décodées en 8 sorties, D0 a D7
- Chaque sortie D*i* représente un des minterms des 3 variables en entrée.
- D*i* = 1 lorsque le nombre binaire xyz = *i*
- En d'autres termes D*i* = m*i*
- Les variables de sortie sont ***mutuellement exclusives***; exactement une sortie a la valeur 1 a tout moment et les autres 7 sorties a 0.

3- vers- 8 lignes Décodeur

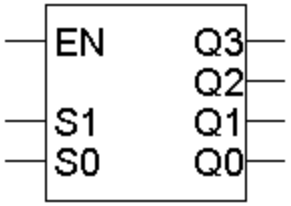


D6 est sélectionne

D6 = 1 autres = 0

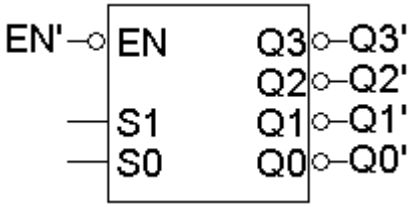
Implémentation d'un décodeur avec des portes NON_ET

- Les décodeurs que avons vu sont dit active-high (i.e. implémentés avec des portes ET)



EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

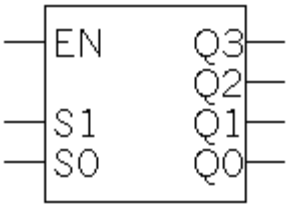
- Les décodeurs Active-low sont implémentés en utilisant les portes NON-ET (i.e. Avec une entrée EN inversé et des sorties inversées).



EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

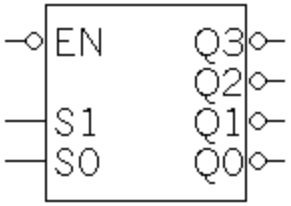
Décodeurs -Active-High et Active-Low

- Comme nous avons vu, un décodeur Active-high génère *minterms*,



$Q3 = S1\ S0$
$Q2 = S1\ S0'$
$Q1 = S1'\ S0$
$Q0 = S1'\ S0'$

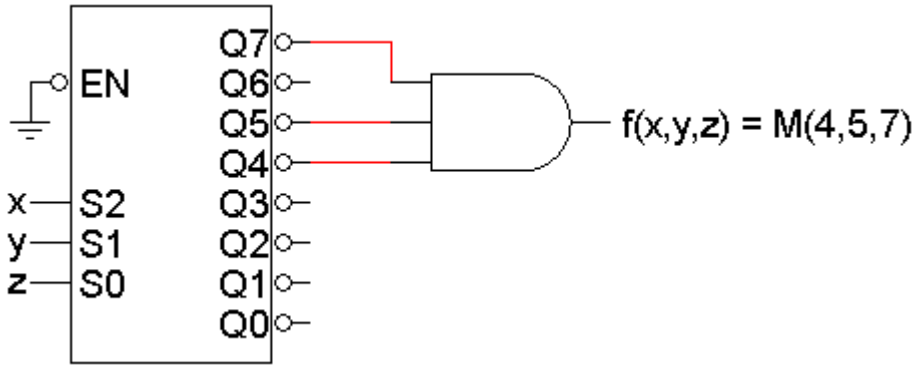
- Un décodeur Active-low génère des *maxterms*.



$Q3' = (S1\ S0)' = S1' + S0'$
$Q2' = (S1\ S0')' = S1' + S0$
$Q1' = (S1'\ S0)' = S1 + S0'$
$Q0' = (S1'\ S0')' = S1 + S0$

Exemple d'un décodeur Active-low

- On peut utiliser un décodeur active-low pour implémenter une fonction produit des maxterms
- Par exemple $f(x,y,z) = \Pi M(4,5,7)$, pet être implémenter avec décodeur active-low comme suit.



- Nous avons besoin d'une porte ET pour faire le produit des sommes.

Réalisation d'une fonction booléenne avec décodeur

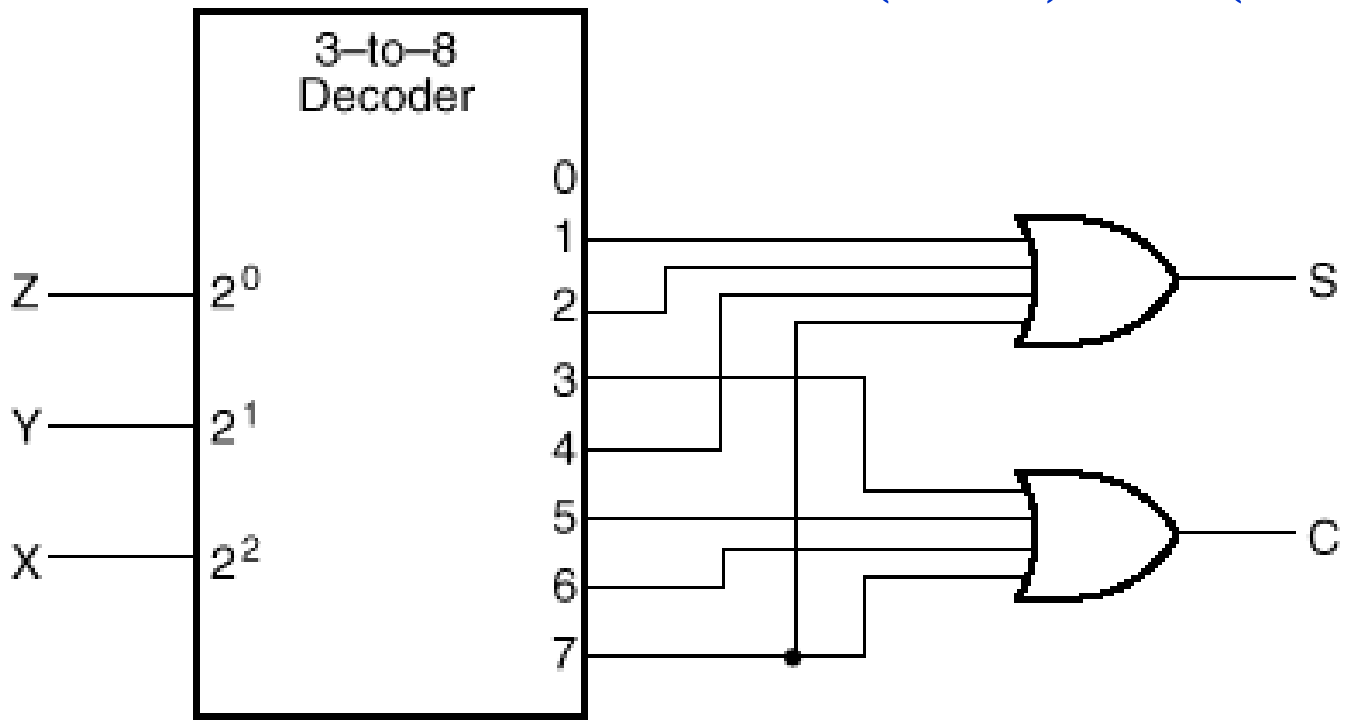
- Tout circuit combinatoire peut être réalise avec des décodeurs et une porte OU! pourquoi?
- Voila un example:
  soit deux fonctions
  - S(X,Y,Z) = **Σ**m(1,2,4,7)
  - C (X,Y,Z) = **Σ**m(3, 5, 6, 7).

→Puisque il y a 3 entrées et un total de 8 minterms, nous avons besoin de 3-a-8 décodeur.

Réalisation d'une fonction booléenne avec décodeur

$S(X,Y,Z) = \Sigma m(1,2,4,7)$

$C(X,Y,Z) = \Sigma m(3,5,6,7)$



Construire un décodeur 3-vers-8 avec deux 2-vers-4

- Une autre manière de concevoir un décodeur de 3-vers 8 est d'utiliser deux décodeurs de 2-vers-4
- A partir de la table de vérité du décodeurs3-vers-8 on peut noter que:
  - Quand S2 = 0, sorties Q0-Q3 sont générées comme un décodeur de 2-vers-4
  - Quand S2 = 1, sorties Q4-Q7 sont générées comme un décodeur de 2-vers-4

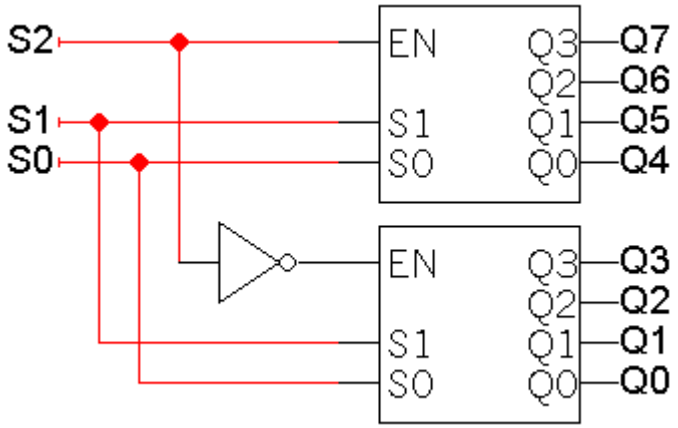
  → S2 peut etre utlisee comme Enable qui active/désactive les deux décodeurs 2-vers-4.

S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Q0 = S2' S1' S0' = m_0
Q1 = S2' S1' S0 = m_1
Q2 = S2' S1 S0' = m_2
Q3 = S2' S1 S0 = m_3
Q4 = S2 S1' S0' = m_4
Q5 = S2 S1' S0 = m_5
Q6 = S2 S1 S0' = m_6
Q7 = S2 S1 S0 = m_7

Construire un décodeur 3-vers-8 avec deux 2-vers-4

- On peut utiliser le enable pour lier les deux décodeurs ensemble. Décodeur 3-to-8 construit avec deux décodeurs 2-vers-4:



S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Décodeur -Expansions

- Des décodeurs plus large peuvent être construits avec plusieurs décodeurs plus petits

→ avec la Conception HIERARCHIQUE

- *Exemple:*
  Un 6-vers-64 décodeur peut être construit avec Quatre 4-vers-16 et UN 2-vers-4 décodeurs:

  →Utiliser le 2-vers-4 décodeur pour générer le signale « enable » pour les quatre 4-vers-16 décodeurs.

4 Entrées décodeurs en arbre

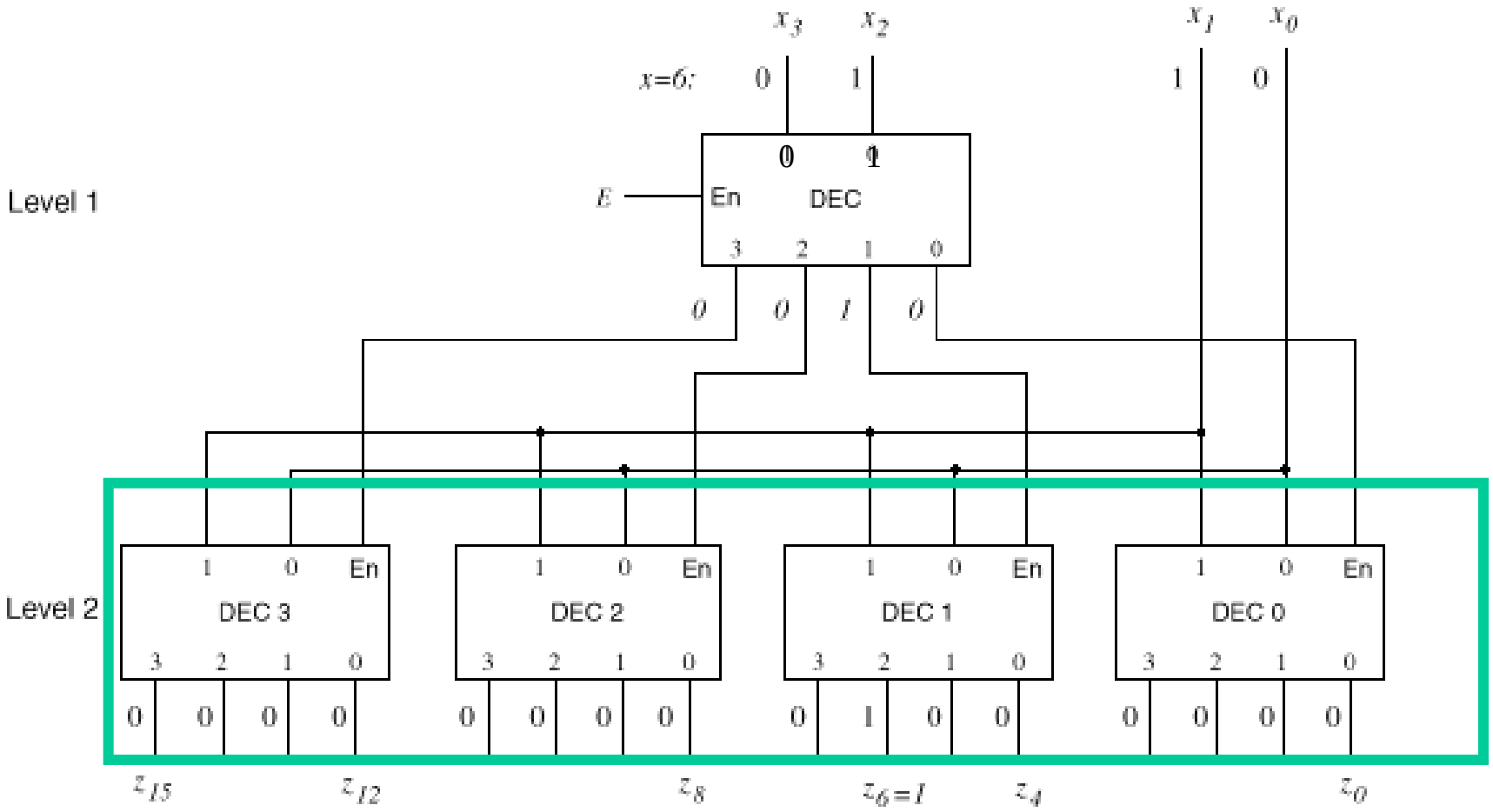


Figure 9.8: 4- nput tree decoder

Encodeur

- Un encodeur possède un nombre de lignes en entrée avec seulement une entrée qui est active a un moment donné
- C'est l'opposé du processus du décodeur

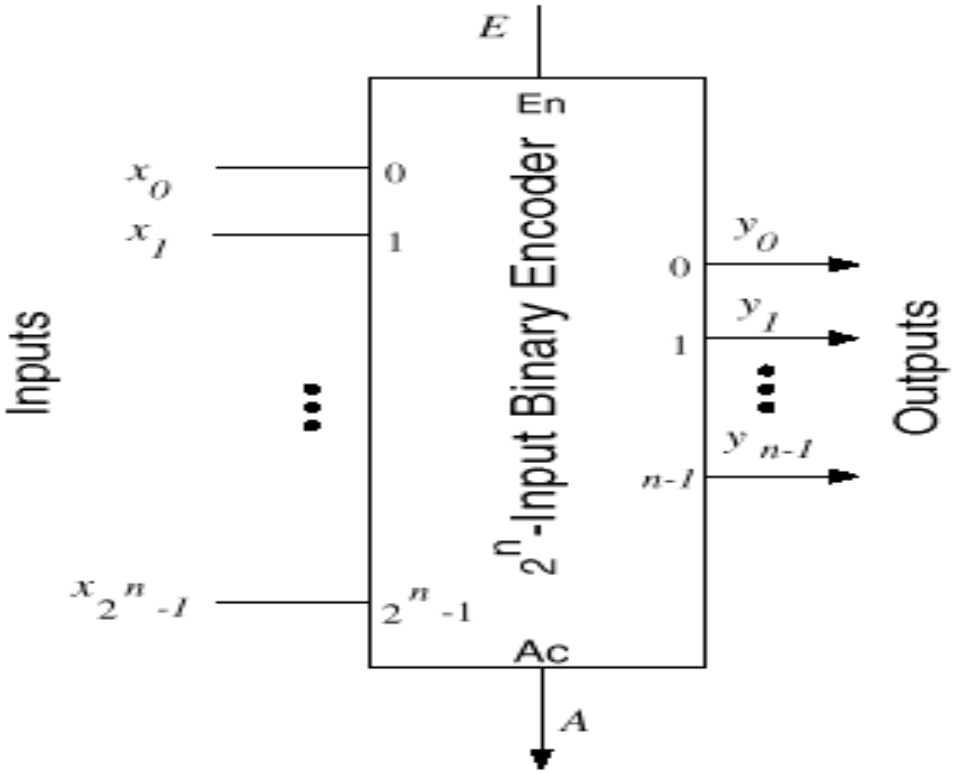


Figure 9.12: 2--input binary encoder.

Encodeur

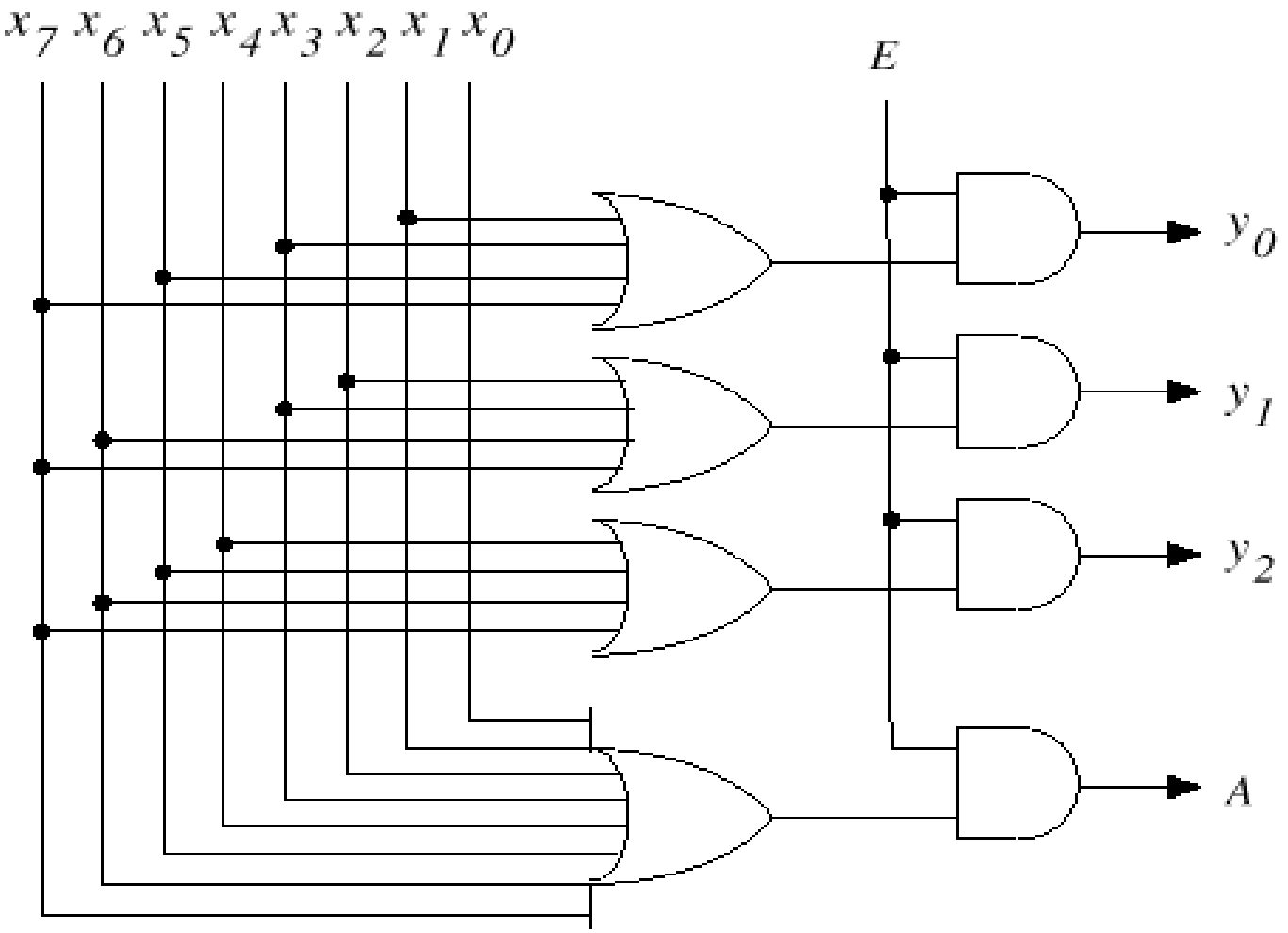


Figure 9.13: mplementation of an 8-input binary encoder.

Encodeur

Exemple: **8-to-3 Encodeur**

D7	D6	D5	D4	D3	D2	D1	D0	Q2	Q1	Q0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Encodeur

•Dans la table de vérité précédente chaque ligne sélectionnée entre 0 et 7 génère son propre code binaire comme par exemple pour 1 c'est 001, 5 c'est 101 et ainsi de suite

•Les fonctions booléennes pour les sorties sont

$$Q2 = D4 + D5 + D6 + D7$$

$$Q1 = D2 + D3 + D6 + D7$$

$$Q0 = D1 + D3 + D5 + D7$$

Problème de conception des encodeurs

- Il y deux ambiguïtés qui sont associées a la conception d'un encodeur simple
  - Seul une entrée peut être activée a un moment donnée . Si deux entrée sont activée simultanément les sorties produisent un résultat indéterminé. (par exemple, si D3 et D6 sont a 1 simultanément, la sortie serait 111.
  - Une sortie avec tout zéro peut être générée quand tous les entrées sont a zéro ou quand D0 =1

Encodeurs a priorité

Encodeur a priorité:

- Permet de résoudre l'ambiguïté.
- Les entrés multiples sont permises mais une possède la priorité sur les autres
- Une indication séparée si les entrées ne sont affectées pas.

Exemple: 4-to-2 Priorité Encodeur Table de Vérité

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

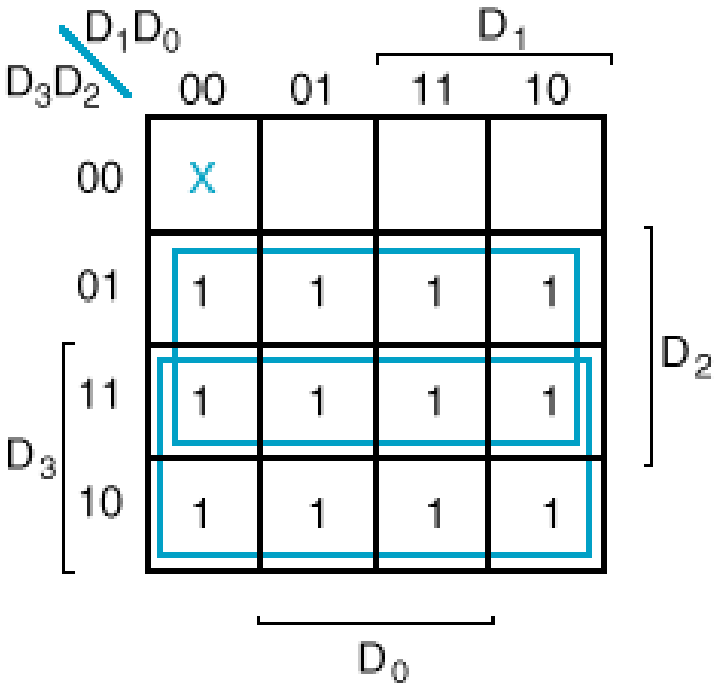
4-to-2 Priorité Encodeur (cont.)

- L’ opération de l’encodeur a priorité est telle que :

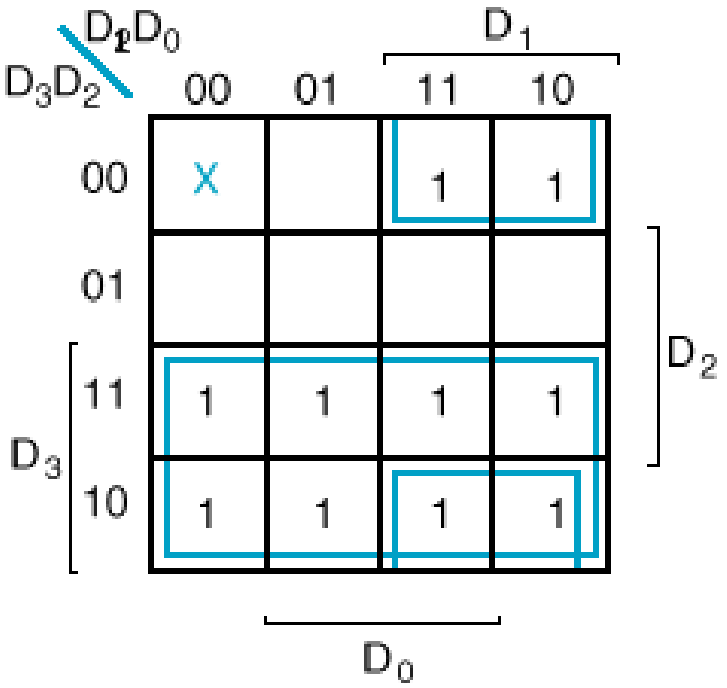
1-si deux ou plus des entrées sont a 1 en même temps, l’entrée qui a le nombre de position le plus élevé prend la priorité.

2- Un indicateur de sortie valide, désigné par V, est mis a 1 seulement quand un ou plus des entrées sont a 1, $V = D_3 + D_2 + D_1 + D_0$ par constatation.

4-to-2 Priorité Encodeur (cont.)

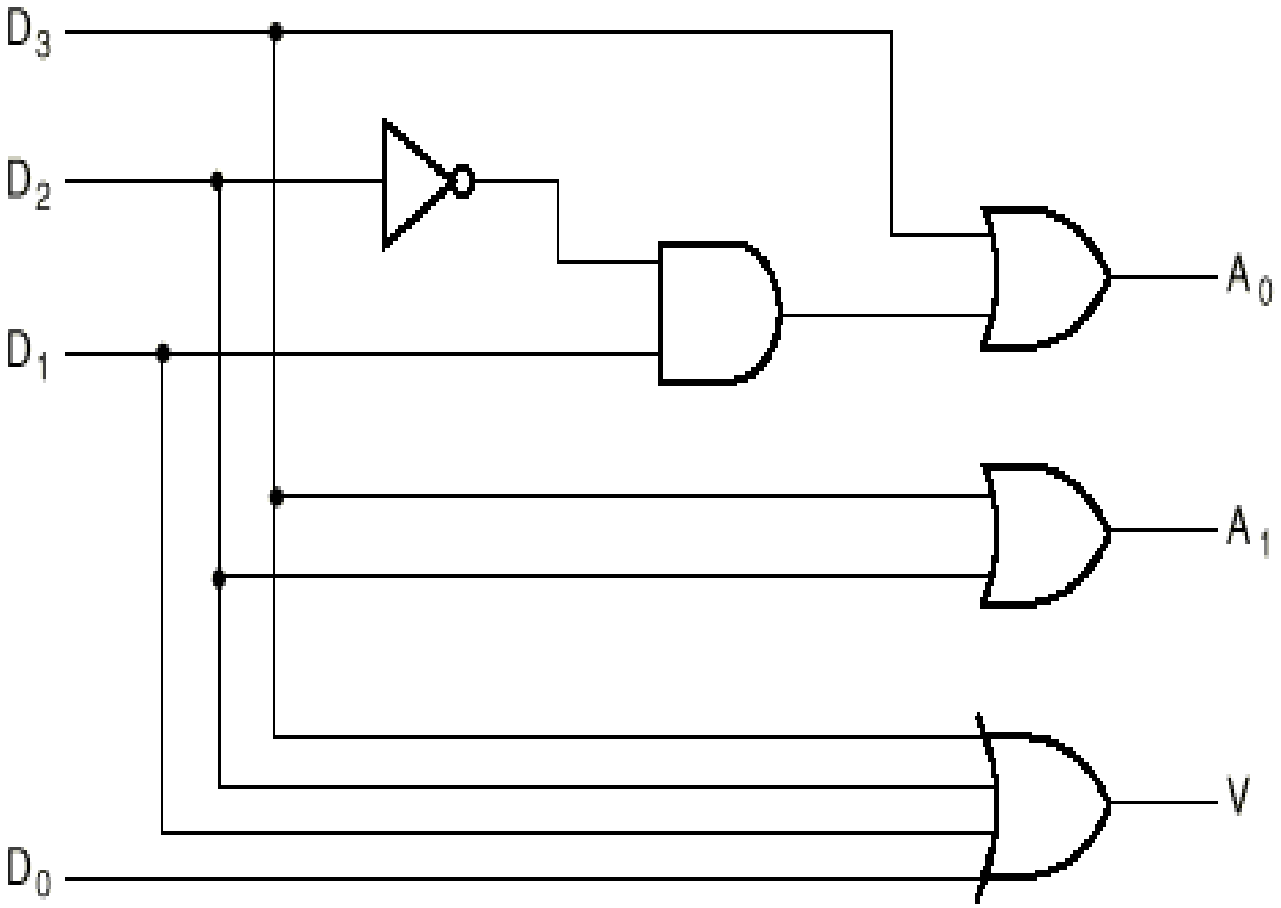


$A_1 = D_2 + D_3$



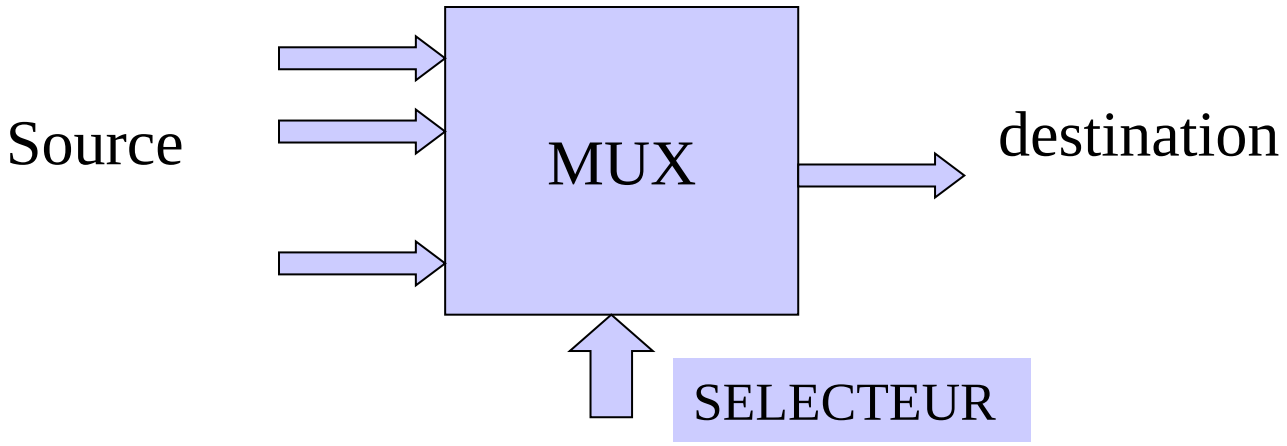
$A_0 = D_3 + D_1\overline{D}_2$

4-to-2 Priority Encoder (cont.)



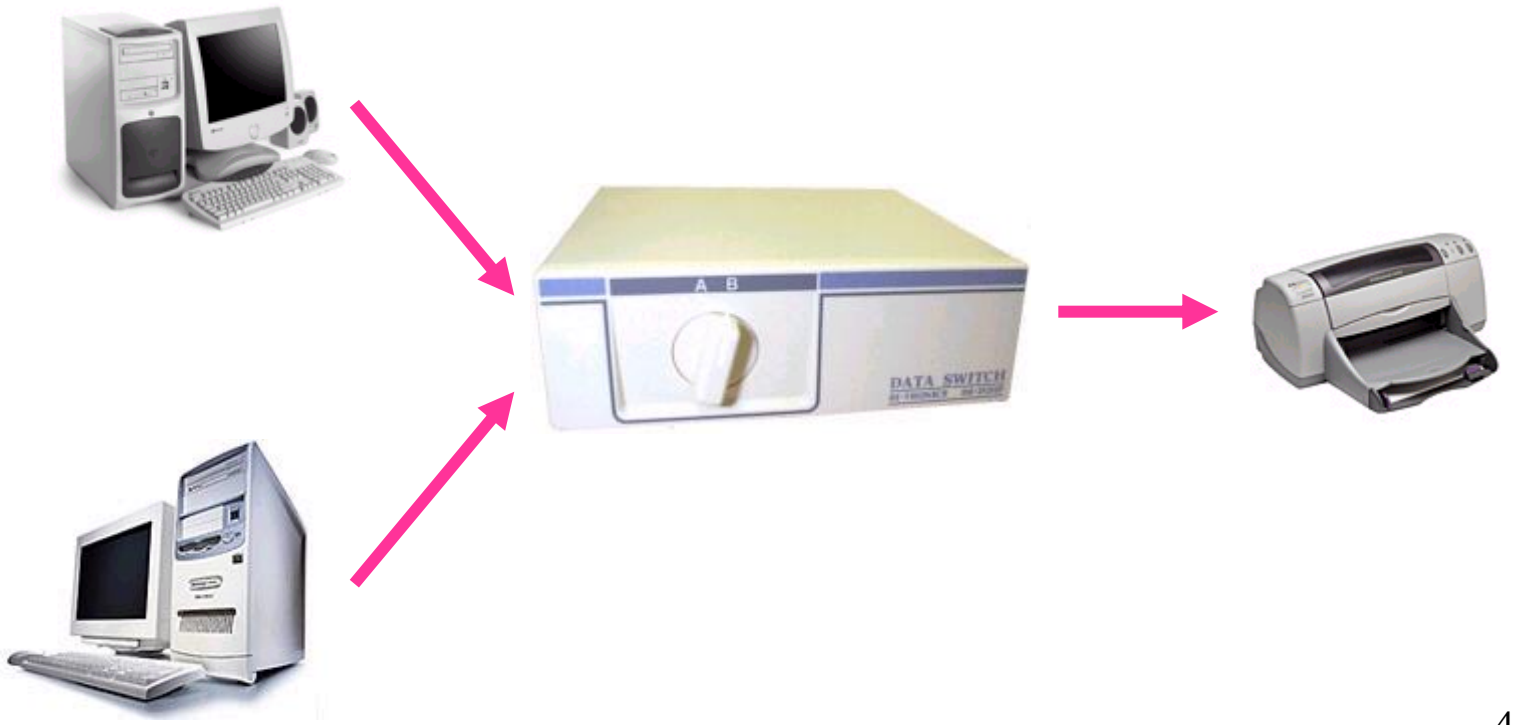
Multiplexeur (MUX): sélecteur de données

- un multiplexeur sélectionne un parmi plusieurs signaux en entrée et le passe vers la sortie.
- La routage de la donnée d'entrée sélectionnée et contrôlée par un sélecteur des entrées.



Multiplexeur

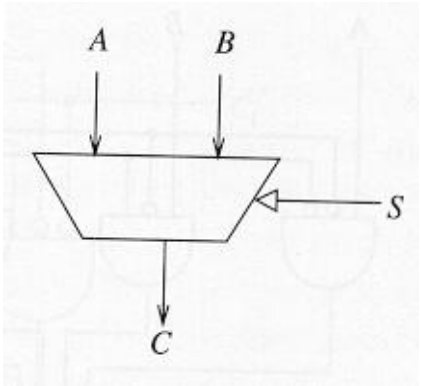
- Multiplexeurs, or « muxes » sont utilisées pour choisir entre les ressources
- Exemple réel : dans le passe avant les réseaux plusieurs ordinateurs peuvent partager une impriment en utilisant un « Switch. »



Multiplexeur (MUX): sélecteur de données

- C'est un circuit combinatoire qui possède 2^n *entrées de donnée*, 1 sortie de donnée et n-bit de *control en entrée* qui sélectionne une entrée de donnée possible.

C prend la valeur de A ou B dépendant de la valeur de S



2-to-1 Multiplexeur

S	A	B	C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

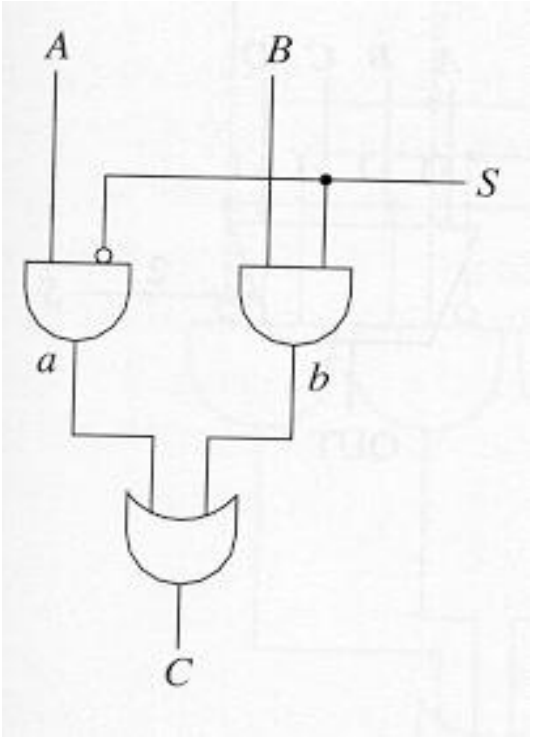
2-vers-1 Multiplexeur

Lorsque S= 0 → C= A, lorsque S= 1 → C = B

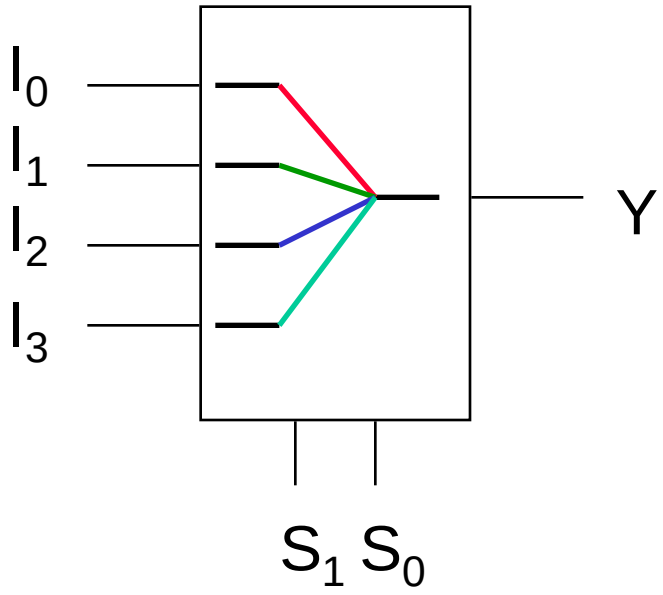
$$C = \bar{S}.A.\bar{B} + \bar{S}.A.B + S.\bar{A}.B + S.A.B$$
$$= \bar{S}.A + S.B$$

Entrée			Sortie
S	**A**	**B**	**C**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Réalisation a deux niveaux



4 vers 1 Multiplexeur



Si $(S_1S_0)_2 = 0$ Alors $Y = I_0$

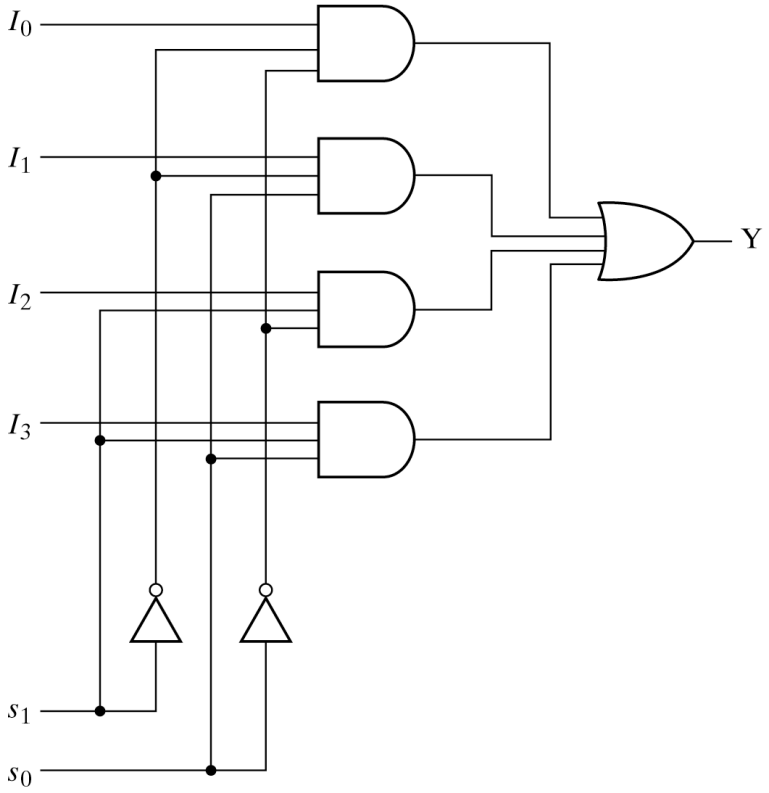
$$Q = \overline{S_0}.\overline{S_1}.I_O$$

Si $(S_1S_0)_2 = 1$ Alors $Y = I_1$

$$Q = S_0.\overline{S_1}.I_1$$

$$Y = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

4 vers 1 MUX- réalisation a deux niveaux



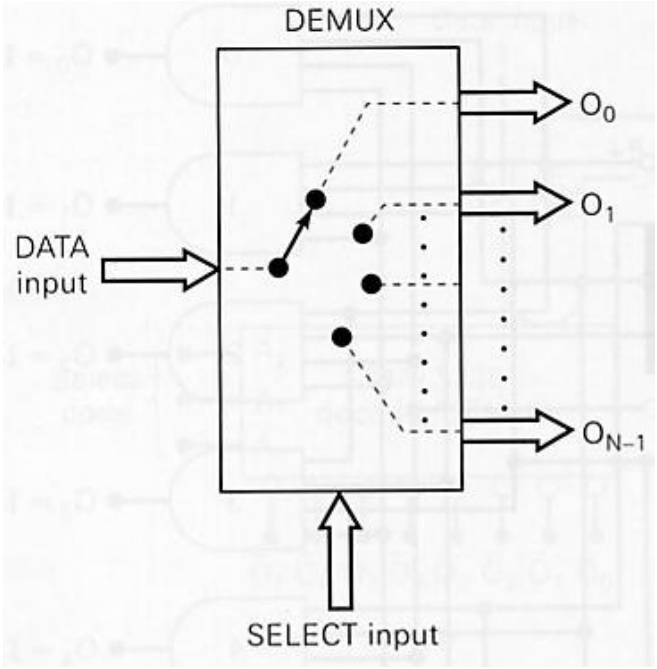
(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Demultiplexeur (DEMUX): Distributeur de données

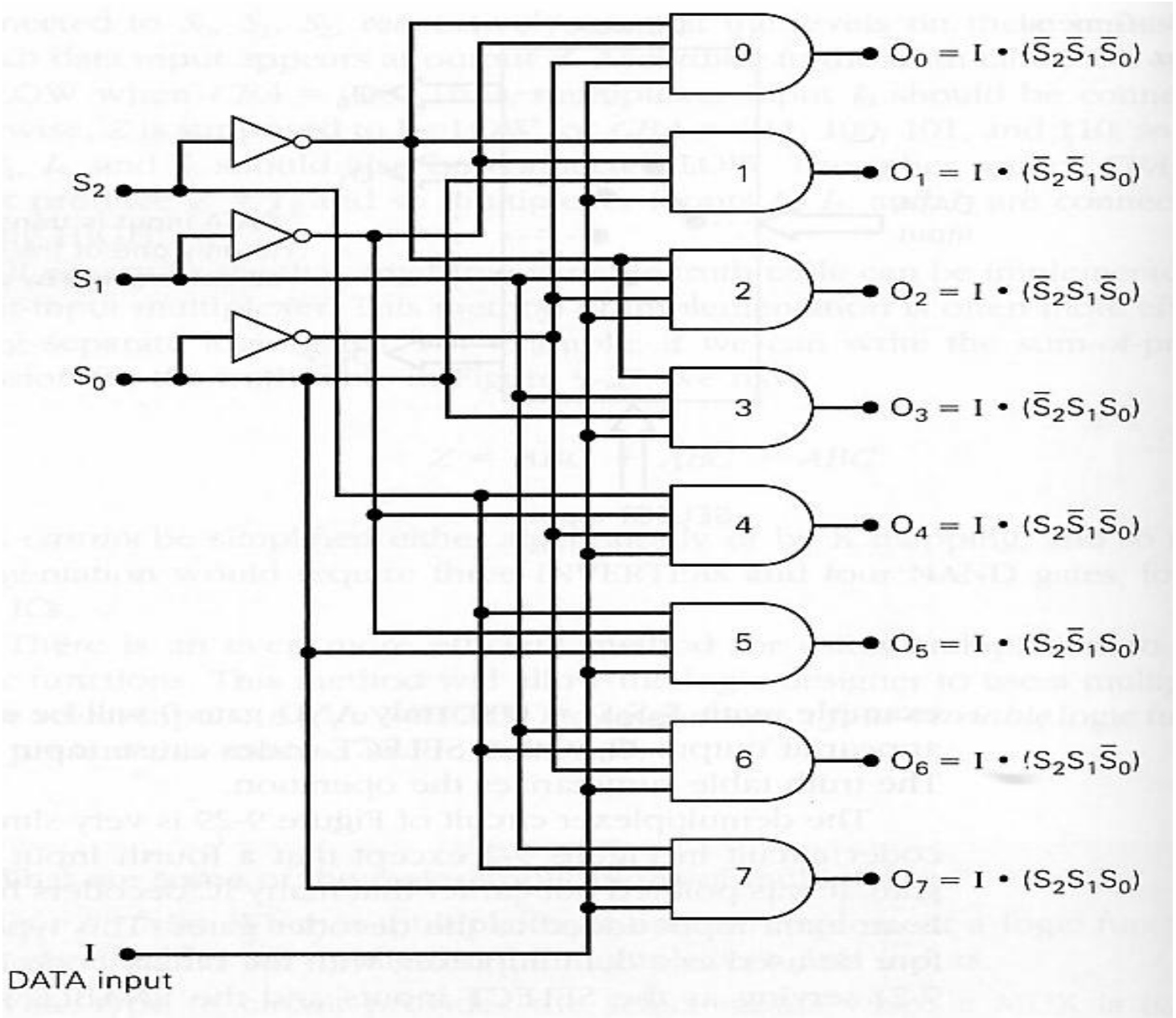
- Demultiplexeur (DEMUX) prends une entre unique et la distribue sur plusieurs sorties



1 ligne vers 8 lignes Demultiplexeur

SELECT code			OUTPUTS							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

1 ligne vers 8 lignes Demultiplexeur



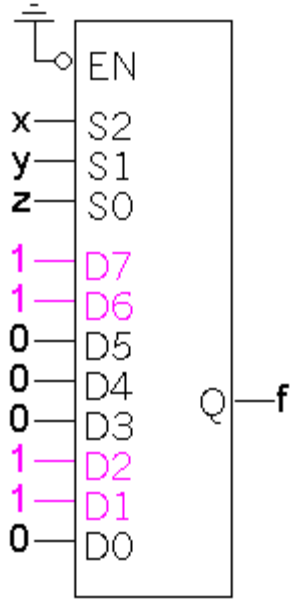
Implémentation de fonctions avec multiplexeurs

- Un multiplexeur peut être utilisé pour implémenter des fonctions arbitraire
- Une manière d'implémenter une fonction a *n* variables est d'utiliser un *n-vers-1* multiplexeur

-Pour chaque minterm m_i de la fonction connecter 1 entrée de donnée (data input) Di. Chaque entrée de donnée correspond a une ligne de la table de vérité

- Connecter les variables d'entrée de la fonction aux entrees des sélecteurs. Exemple, $f(x,y,z) = \Sigma m(1,2,6,7)$ peut être implémenter comme suit

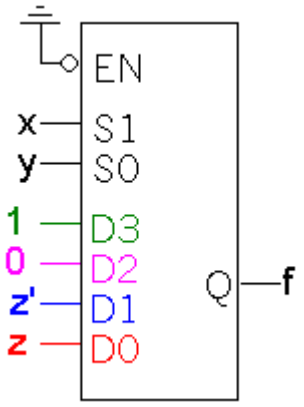
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Implémentation simplifiée (plus efficace)

- On peut implémenter $f(x,y,z) = \Sigma m(1,2,6,7)$ avec juste un 4-vers-1 multiplexeur, au lieu de 8-vers-1.
- Etape 1: trouver la table de vérité pour la fonction, et grouper les lignes en pair. Dans chaque pair de ligne x et y sont les mêmes , donc f est fonction de of z seulement
  - Quand xy=00, f=z
  - Quand xy=01, f=z'
  - Quand xy=10, f=0
  - Quand xy=11, f=1
- Etape 2: connecter les deux première variables de table de vérité (ici, x et y) au bits du sélecteur S1 S0 de 4-vers -1 mux.
- Etape 3: Connecter les équations ci-dessus pour f(z) to aux entrée de donné D0-D3.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Examples

- Implémenter la fonction suivante avec un multiplexeur
  - $F(A,B,C,D) = \Sigma m\ (1,3,4,11,\ 12,13,\ 14,15)$

Exemple (suite)

- f(A,B,C,D) = Σm (1,3,4,11, 12,13, 14,15) n-1 selection avec 2 puissance n-1

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

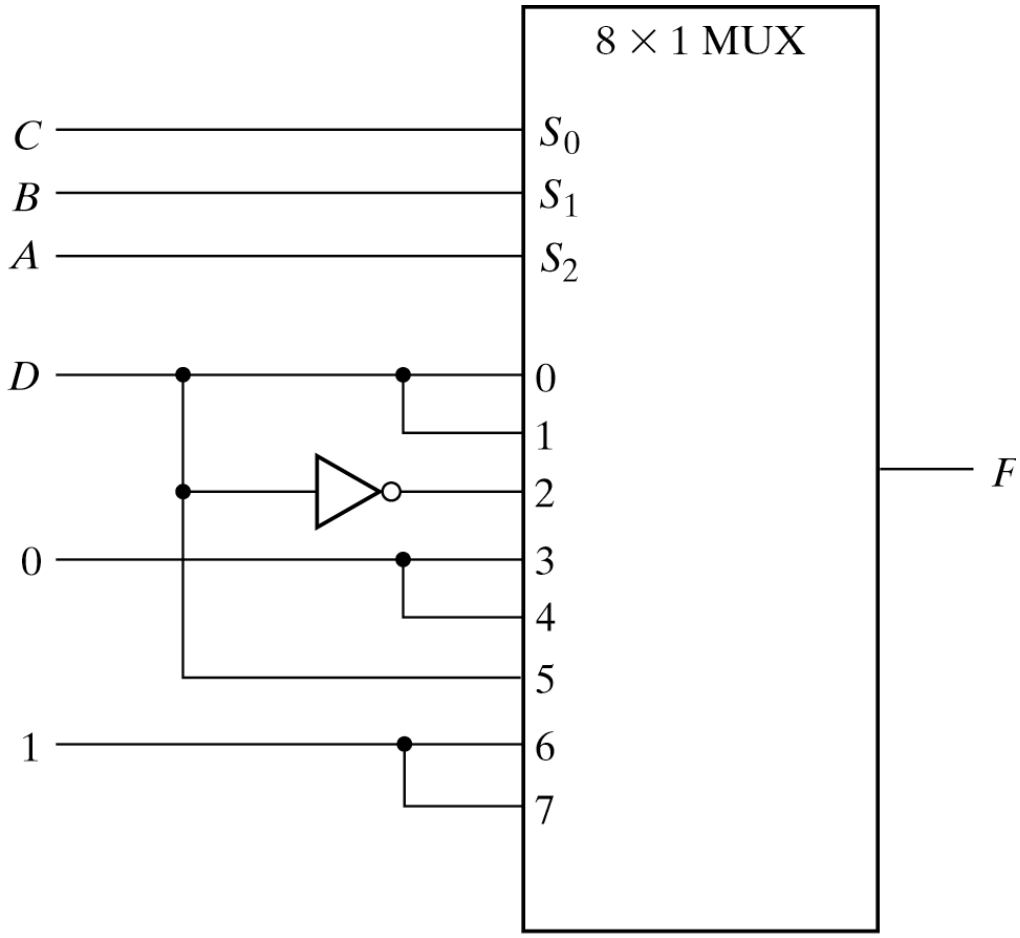


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Autres exemples de circuits combinatoires

Les exemples suivants seront traités dans le DGD

- Comparateur a 4 bits
- Multiplicateur binaire 4 bits par 3 bit
- Additionneur décimal

Analyse des circuits logiques

→ Le problème: A partir d'un circuit donnée comment reconstruire la fonction logique?

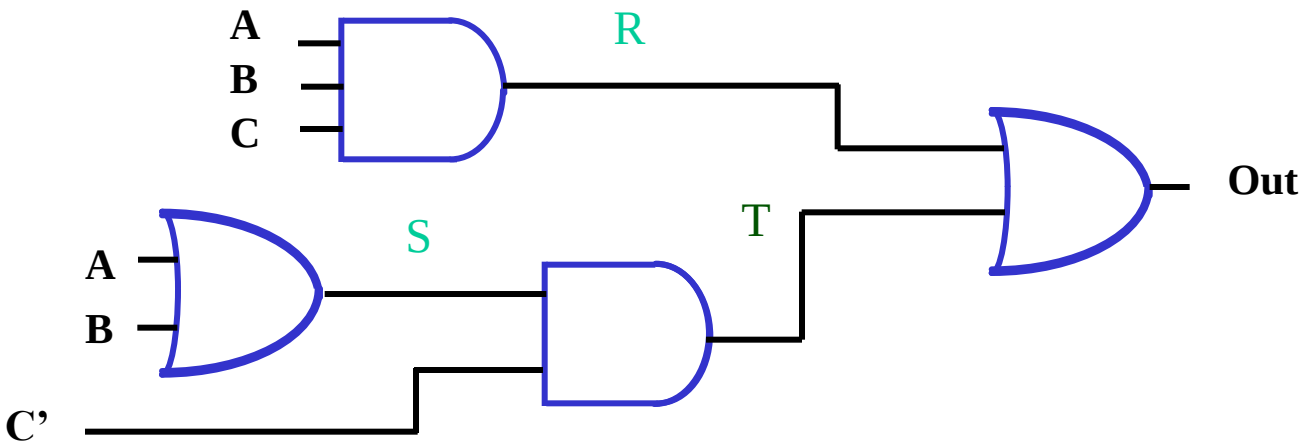
- Analyse des circuits logiques
  - Étant donnée un circuit
    - Créer la table de vérité
    - Créer le circuit simplifié
- Approches
  - expression Booléenne
  - table de vérité

Donner un nom aux sorties des portes

- **Donner un nom aux sorties des portes qui sont fonction des variables d'entrées.**

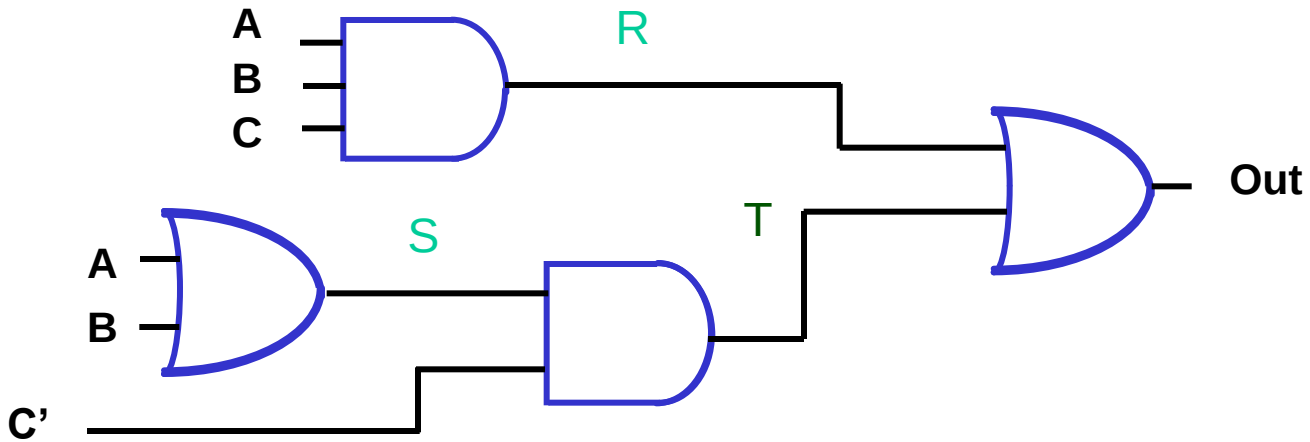
- **Donner un nom aux sorties des portes qui sont fonction des variables d'entrées et des soties de portes déjà nommées**

- **Recommencer jusqu'à ce que toutes les sorties des portes sont nommées.**



Approche 1: Créer des équations intermédiaires

- **Étape 1: Créer une équation pour chaque sortie de porte en fonction de ses entrées**
  - **R = ABC**
  - **S = A + B**
  - **T = C'S**
  - **Out = R + T**

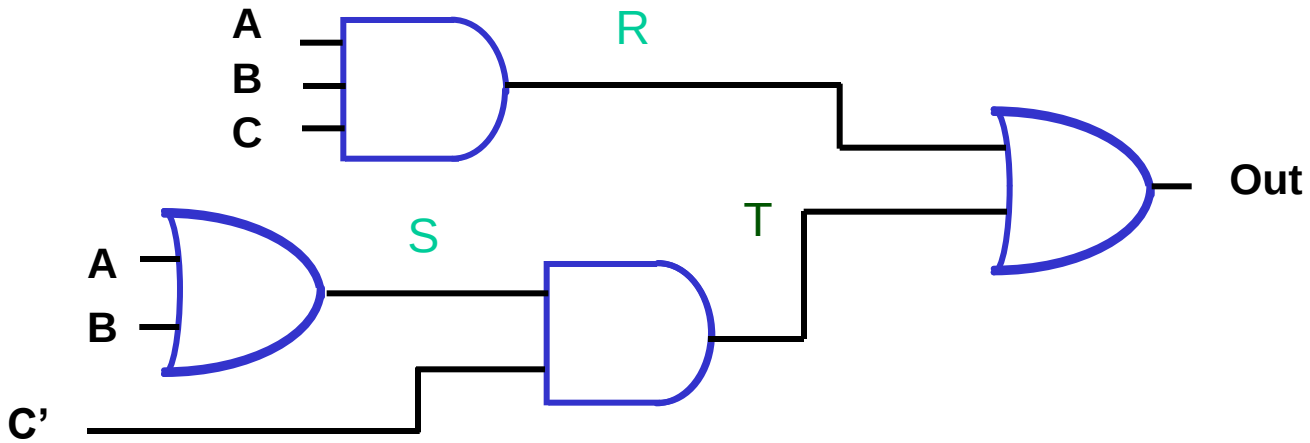


Approche 1: Substitutions

- **Étape 2: effectuer la substitution par des variables d'entrées**

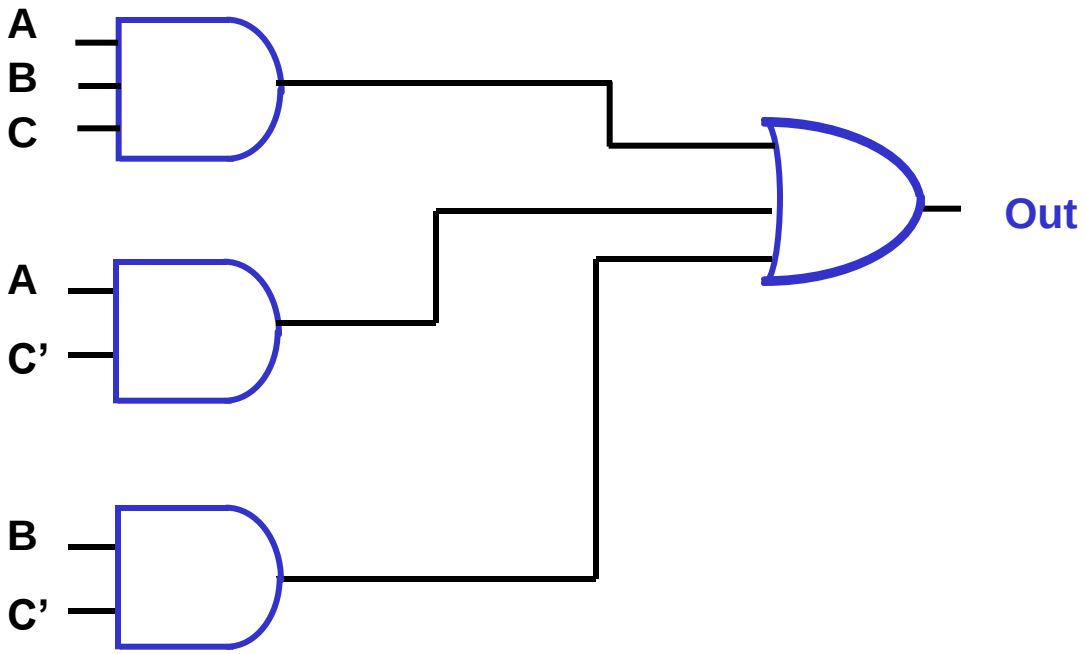
(A, B, C)

- **R = ABC**
- **S = A + B**
- **T = C'S = C'(A + B)**
- **Out = R + T = ABC + C'(A+B)**



Approach 1: Substitution

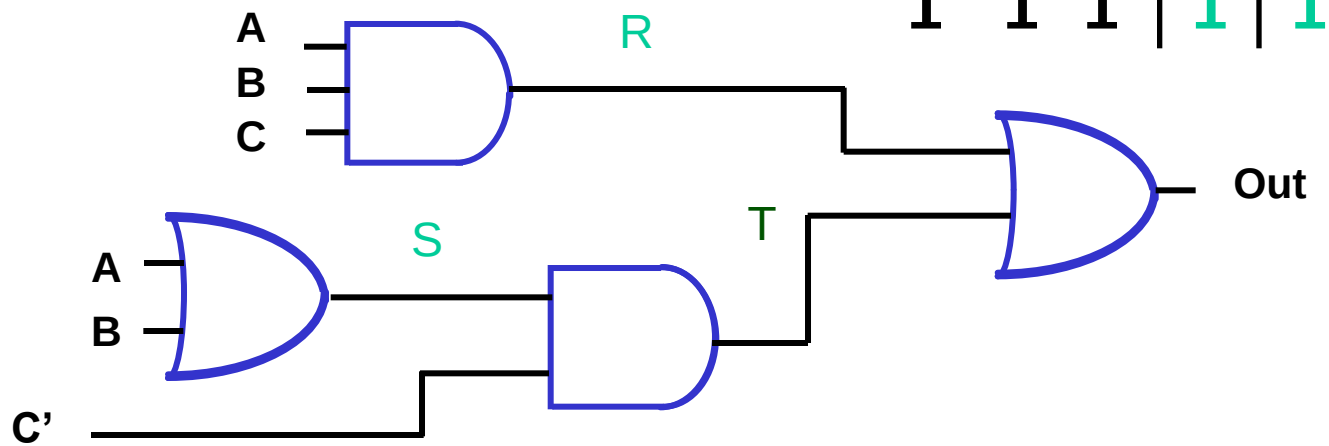
- **Étape 3: obtenir l'équation finale somme de produits**
  - **Out = ABC + C'(A+B) = ABC + AC' + BC'**



Approche 2: Table de Vérité

- **Étapes 1: Déterminer les sorties pour les fonctions intermédiaires a partir des variable d'entrées**

A	B	C	R	S
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

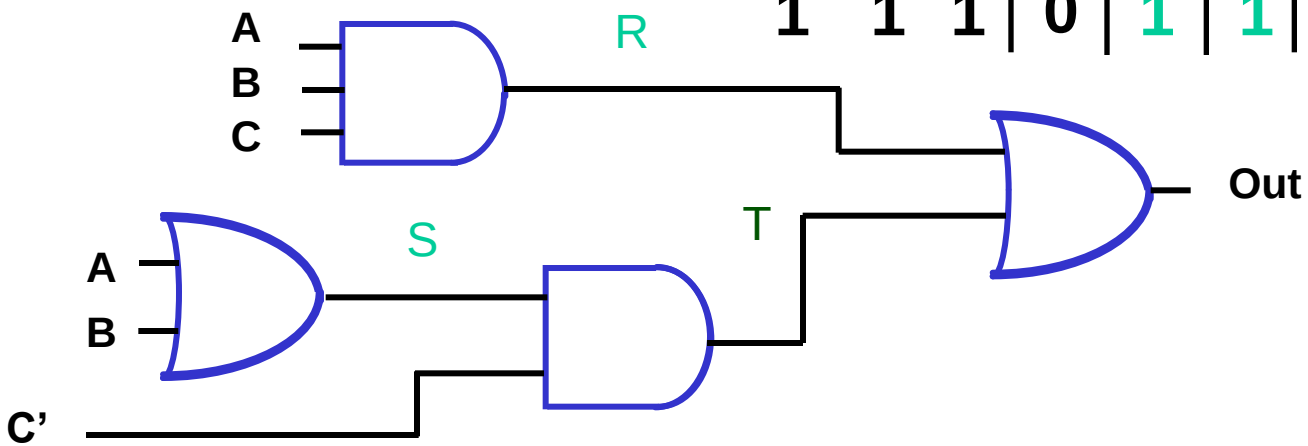


Approche 2: Table de Vérité

- **Étape 2: évaluer les fonctions intermédiaires**

T = S * C’

A	B	C	C’	R	S	T
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	1	1	0

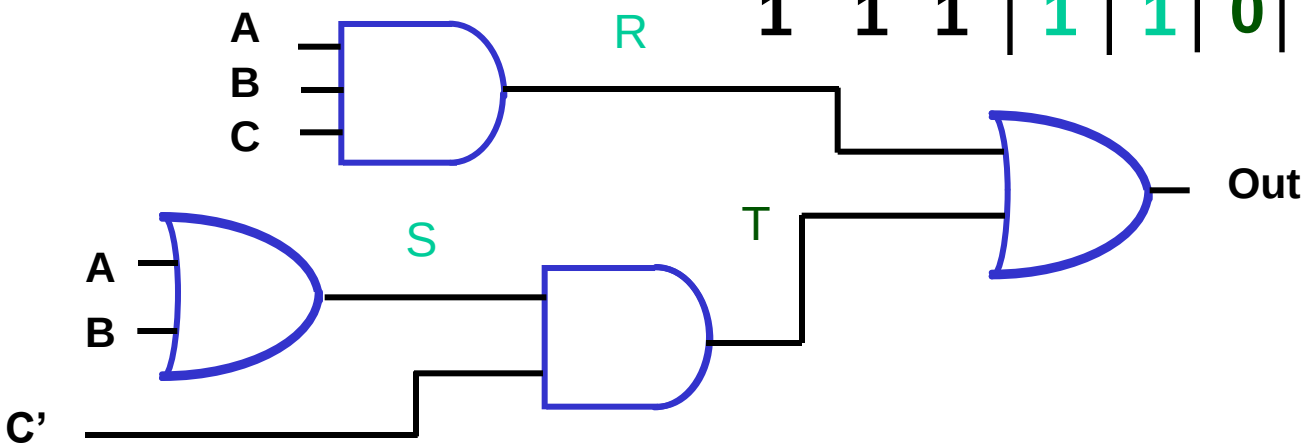


Approche 2: Table de Vérité

- **Step 3: Determiner la sortie pour la fonction.**

R + T = Out

A	B	C	R	S	T	Out
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	1	0	1



Autre exemple

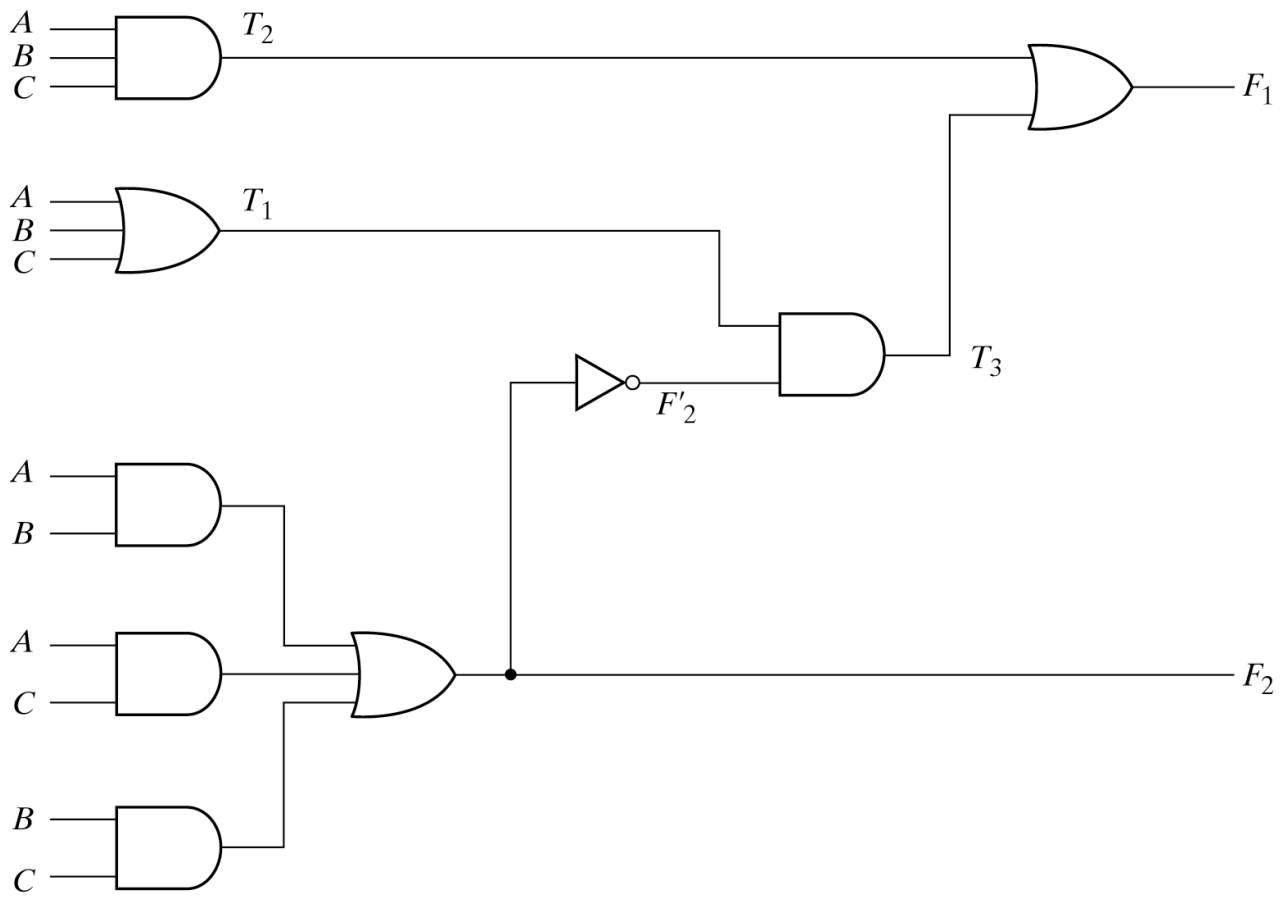


Fig. 4-2 Logic Diagram for Analysis Example

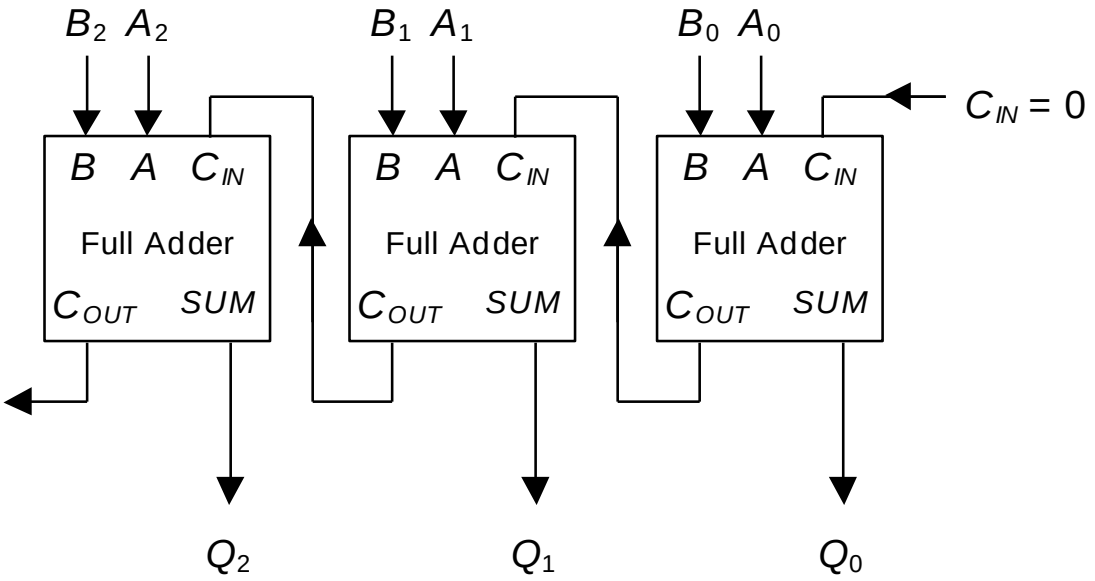
Approche 2: Table de Vérité

- **La fonction peut être simplifie en utilisant le diagramme de Karnaugh**

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

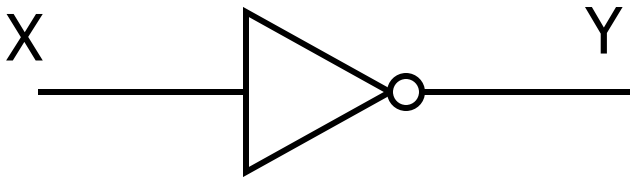
Additionneur Parallèle

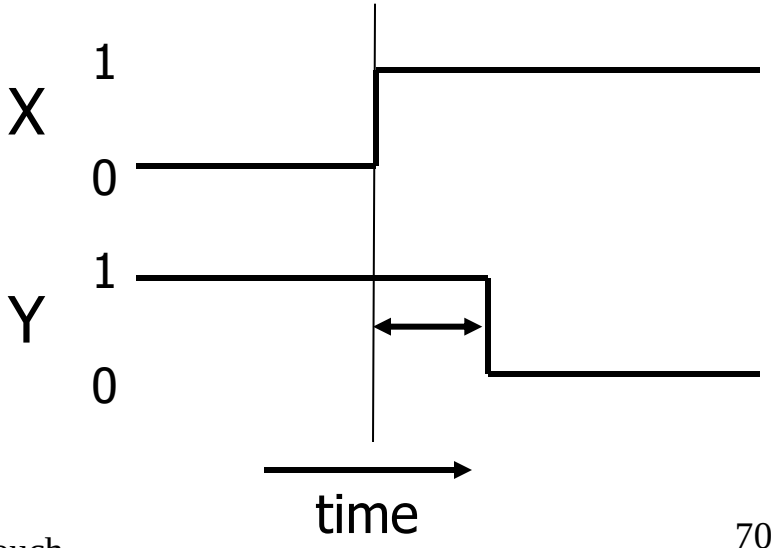
- Rappel: pour additionner deux nombres binaires de *n*-bit *n* additionneurs complet (full-adder) doivent être connectes en cascade.
- Chaque additionneur représente une colonne dans l'addition
- La retenue (le signal) se propage au travers les additionneurs de la droite vers la gauche.



Délai de Propagation

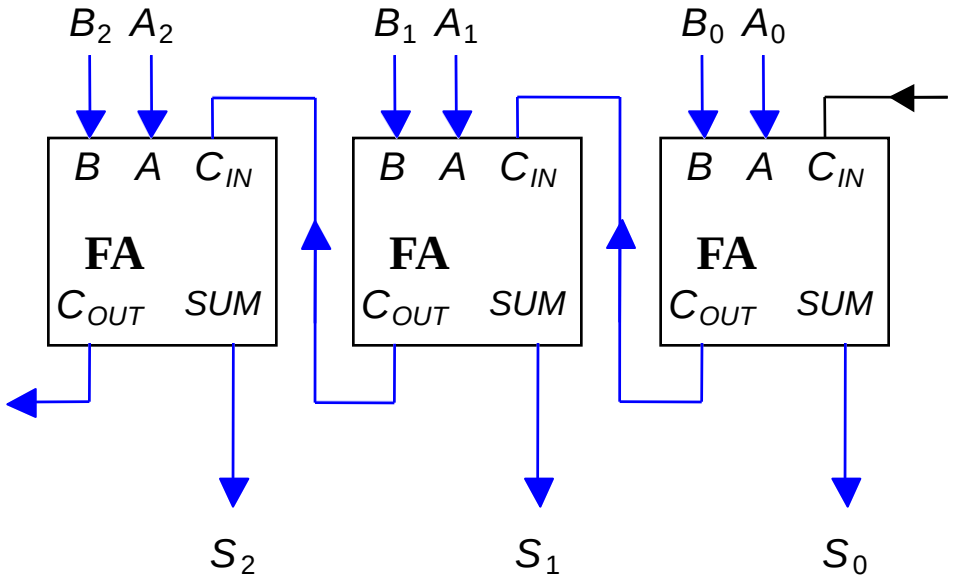
- Toutes les portes logiques prennent un délai de temps non nul pour répondre a un changement dans les entrées.
- De délai représente le *temps de propagation* par la porte logique qui est typiquement mesure en dizaine de nanosecondes.





Carry Ripple

- *Les entrées A* et *B* change, les changements correspondants a C_{IN} sont propagées au travers du circuit.



t = 0, A & B change

t = 30 ns, Adder 0 les sorties repondent

t = 60 ns, Adder 1 les sorties repondent

t = 90 ns, Adder 2 les sorties repondent

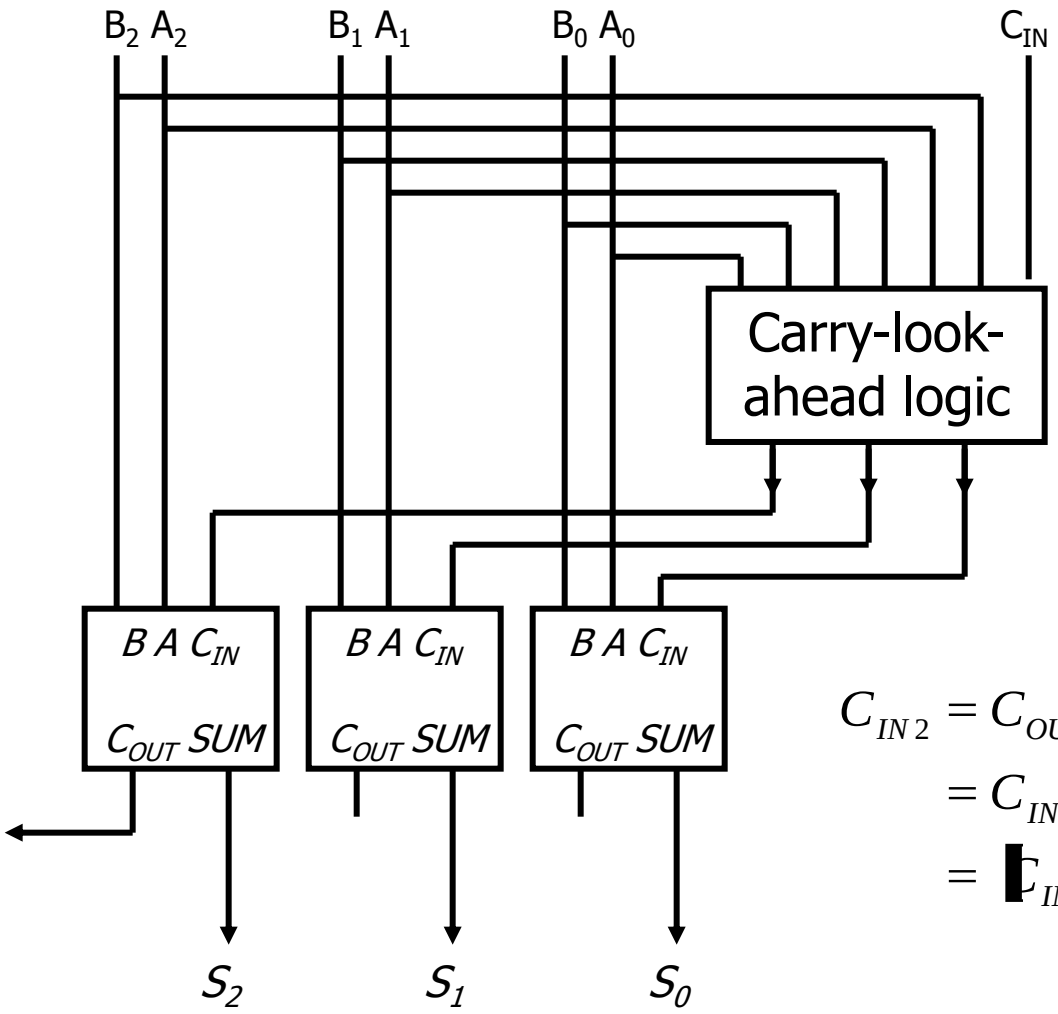
Retenue Anticipée

- Le délai accumulé dans des additionneurs parallèles peuvent être très grand.
- Exemple : 16 bits utilisant 30 ns Additionneurs Complets:

$$16 \times 30 \text{ ns} = 480 \text{ ns}$$

- Solution : Générer les signaux des retenues directement a partir des entrées *A* et *B* plutôt que d'utiliser la propagation au travers du circuit.

Additionneur à Retenue Anticipée



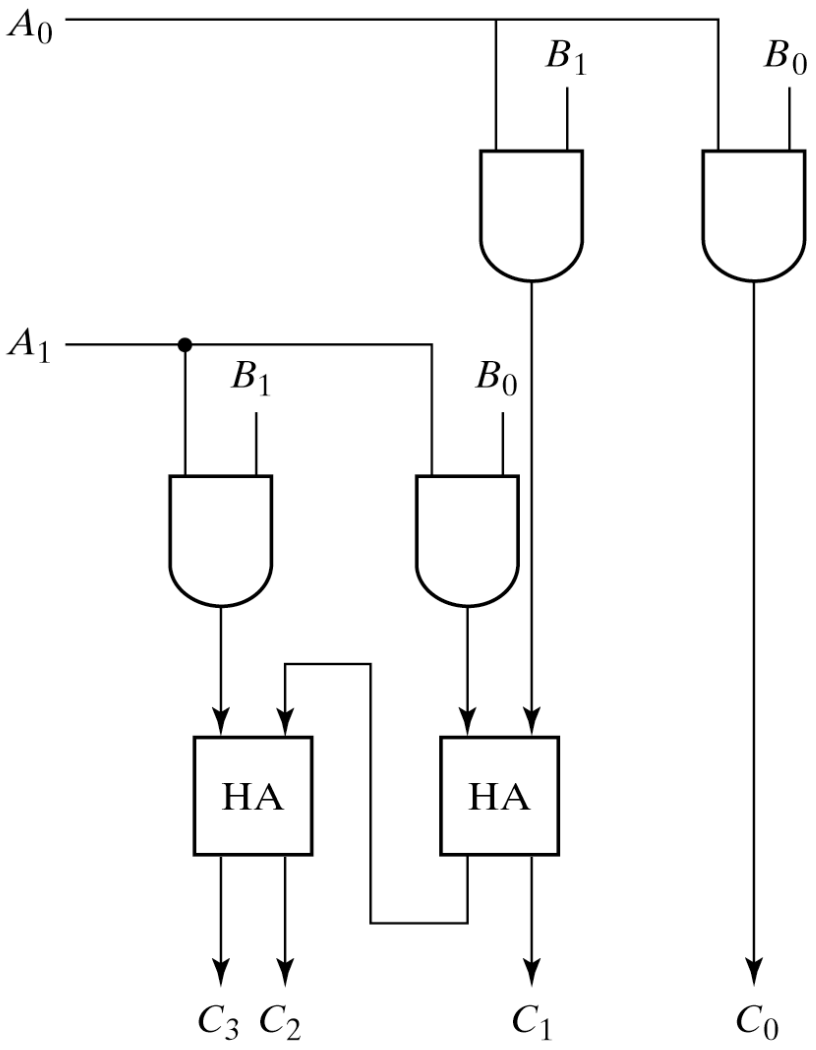
$$C_{IN\,0} = C_{IN}$$

$$C_{IN\,1} = C_{OUT\,0} = C_{IN}(A_0 + B_0) + A_0 B_0$$

$$C_{IN\,2} = C_{OUT\,1} = C_{IN\,1}(A_1 + B_1) + A_1 B_1 = C_{IN}(A_0 + B_0) + A_0 B_0 (A_1 + B_1) + A_1 B_1$$

Circuit de Multiplication binaire

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



- La porte logique ET permet d'obtenir la produit de deux bits.
- Pour un multiplicateur de 2-bit par 2-bit, on peut utiliser deux demi additionneurs pour additionner les produits partiels

- Ici C3-C0 son des produits pas des retenus.

Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4-bit by 3-bit binary multiplier

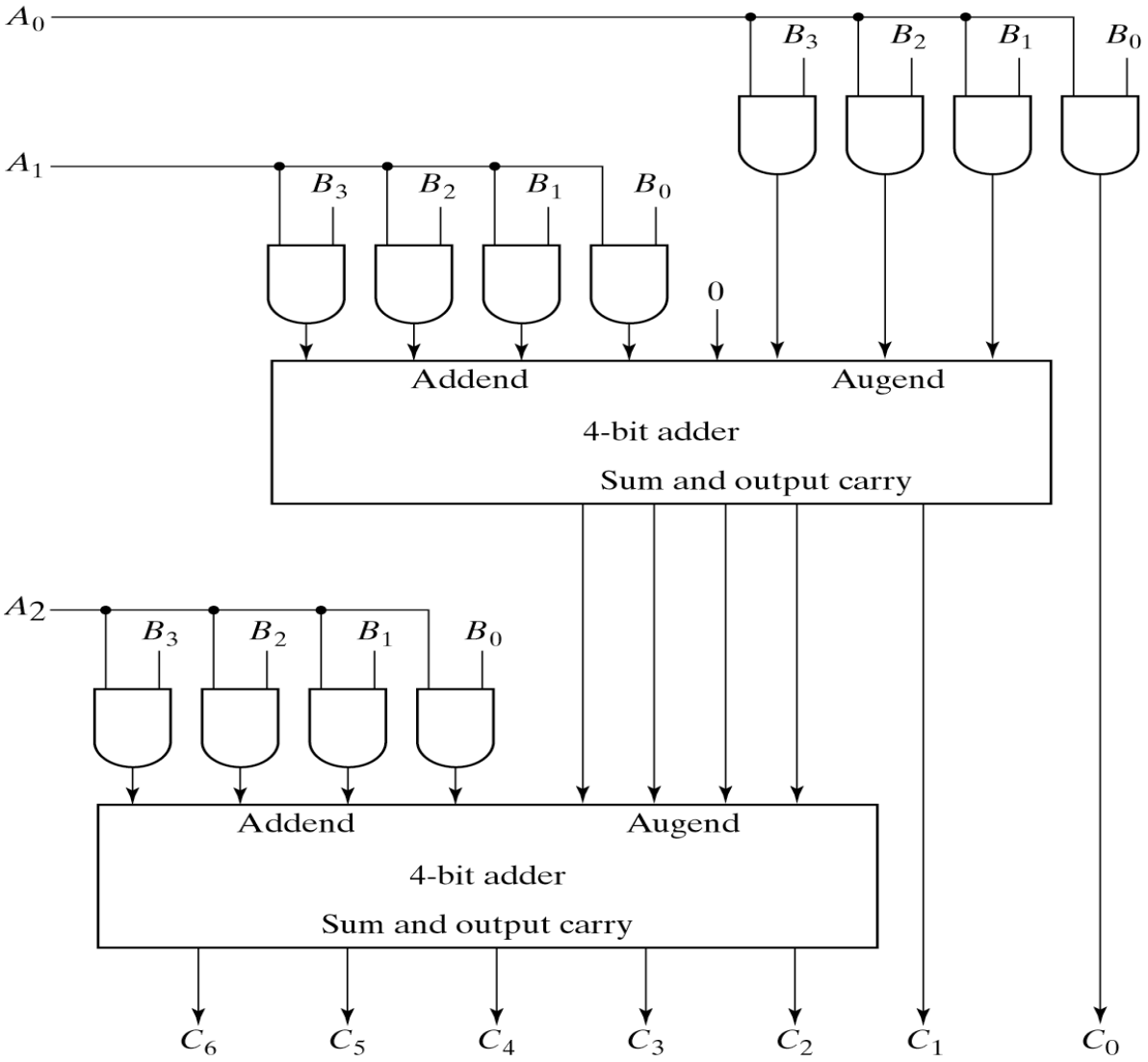


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier