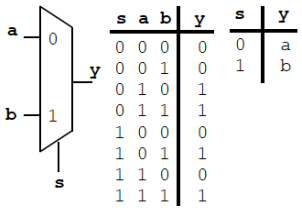
Note de COURS
5 : Circuits combinatoires

Multiplexeurs (MUX)

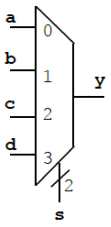
- Circuit logique qui permet de sélectionner un signal d'entrée et de l'envoyer vers la ligne de sortie
- Les équations booléennes des circuits MUX 2-vers-1, MUX 4-vers-1 et MUX 8-vers-1 sont :



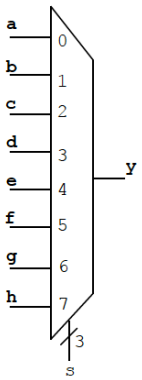
s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

s	y
0	a
1	b

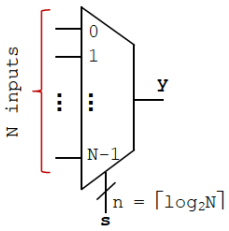
$$y = \overline{s}a + sb$$



$$y = \overline{s_1}\overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$



$$y = \overline{s_2}\overline{s_1}\overline{s_0}a + \overline{s_2}\overline{s_1}s_0b + \overline{s_2}s_1\overline{s_0}c + \overline{s_2}s_1s_0d + s_2\overline{s_1}\overline{s_0}e + s_2\overline{s_1}s_0f + s_2s_1\overline{s_0}g + s_2s_1s_0h$$



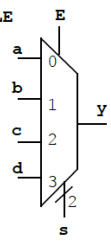
- Dans le cas général, un multiplexeur a $N=2^n$ entrée, 1 sortie et un sélectionneur à n bits. Cependant, si le multiplexeur a N entrée, avec N qui n'est pas une puissance de 2, alors le sélectionneur aura $\lceil \log_2 N \rceil$ bits.

Multiplexeur avec activateur

- Un control supplémentaire peut être ajouté avec un signal d'activation. Si le multiplexeur est activé alors le circuit fonctionne, sinon aucune sortie n'est fournie. Le signal d'activation peut être soit *active-high* (E=1) ou *active-low* (E=0).

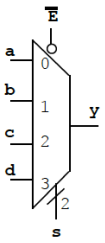
ACTIVE HIGH ENABLE

E	s_1	s_0	y
1	0	0	a
1	0	1	b
1	1	0	c
1	1	1	d
0	X	X	0



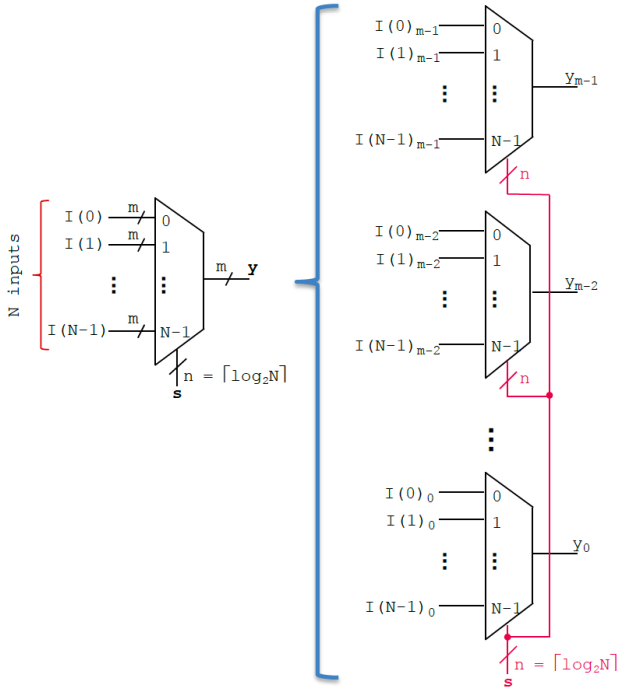
ACTIVE LOW ENABLE

$\overline{E}$	s_1	s_0	y
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	X	X	0



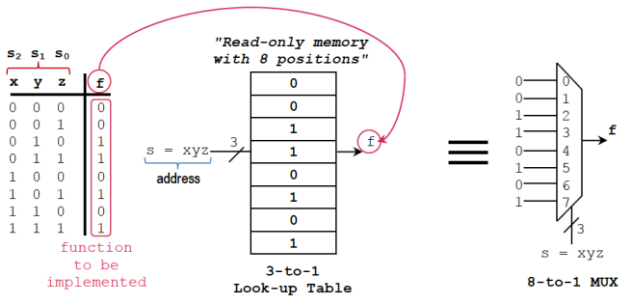
Bus multiplexé

- Généralement on souhaite avoir un signal d'entré sur plus d'un bit.
- Dans ce dessin, chaque signal d'entré contient '*m*' bits.
- Un bus multiplexé peut donc être construit avec '*m*' multiplexeurs, chacun ayant en charge 1 bit d'entrée



Circuit logique à base de MUX

- Les MUX peuvent être utilisés pour implémenter des fonctions booléennes. Le sélectionneur peut être vu comme les variables d'entrées, les bits d'entrés comme des valeurs fixes passées en sortie en fonction de la valeur prise par le sélectionneur.
- Un MUX avec des entrées fixes implémente une fonction logique. Cette fonctionnalité est proche des LUT (*Look-Up Table*) rencontrés dans les FPGA.
- Exemple de LUT 3-vers-1 (LUT à 3 entrées qui contient 2^3=8 adresses)



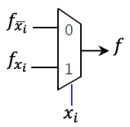
Expansion de Shannon

- Utile pour exprimer une fonction booléenne à l'aide de MUX.
- Une fonction booléenne à *n* variables peut se décomposer en des fonctions booléennes à (*n-1*) variables

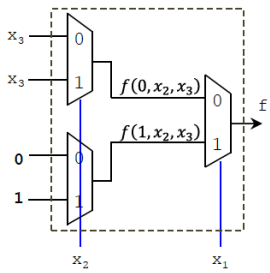
$$f(x_1, x_2, \dots, x_n) = \overline{x_1} f(0, x_2, \dots, x_n) + x_1 f(1, x_2, \dots, x_n)$$

- Dans l'équation précédente on a utilisé une décomposition en fonction de x_1, mais n'importe qu'elle variable peut convenir.
- La notation de l'expansion de Shannon est :

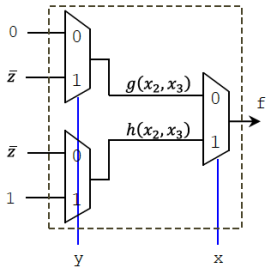
$$f = \overline{x_i} f_{\overline{x_i}} + x_i f_{x_i}$$



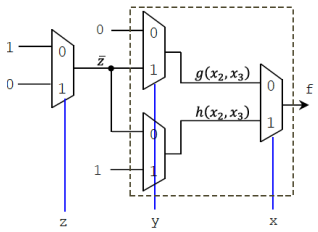
- **Exemples** :
  1. $f = x_1 x_2 + \overline{x_1} x_3 + x_2 x_3$
     $f = \overline{x_1} f(0, x_2, x_3) + x_1 f(1, x_2, x_3) = \overline{x_1}(x_3 + x_2 x_3) + x_1(x_2 + x_2 x_3)$
     Il est possible d'appliquer l'expansion de Shannon aux fonctions à 2 variables :
     $f = \overline{x_1} g(x_2, x_3) + x_1 h(x_2, x_3)$
     $g(x_2, x_3) = \overline{x_2} g(0, x_3) + x_2 g(1, x_3) = \overline{x_2}(x_3) + x_2(x_3)$
     $h(x_2, x_3) = \overline{x_2} h(0, x_3) + x_2 h(1, x_3) = \overline{x_2}(0) + x_2(1)$



2. $f = \bar{z}y + \bar{z}x + xyz$
$f = \bar{x}f(0,y,z) + xf(1,y,z) = \bar{x}(\bar{z}y) + x(\bar{z}y + \bar{z} + yz)$
$f = \bar{x}g(y,z) + xh(y,z)$
$g(y,z) = \bar{z}y = \bar{y}g(0,z) + yg(1,z) = \bar{y}(0) + y(\bar{z})$
$h(y,z) = \bar{z}y + \bar{z} + yz = \bar{y}h(0,z) + yh(1,z) = \bar{y}(\bar{z}) + y(1)$

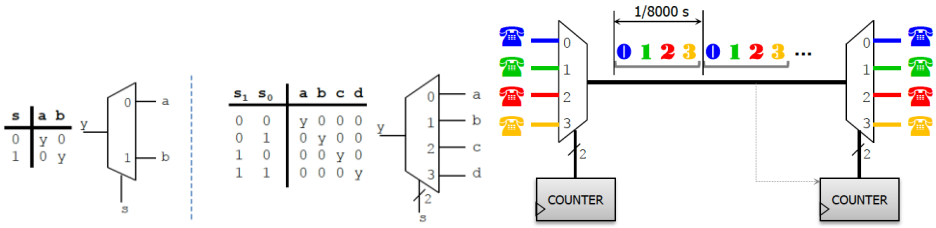


Il est possible d'implémenter $\bar{z}$ à l'aide de MUX :
$p(z) = \bar{z} = \bar{z}p(0) + zp(1) = \bar{z}(1) + z(0)$
Ce qui donne au final :



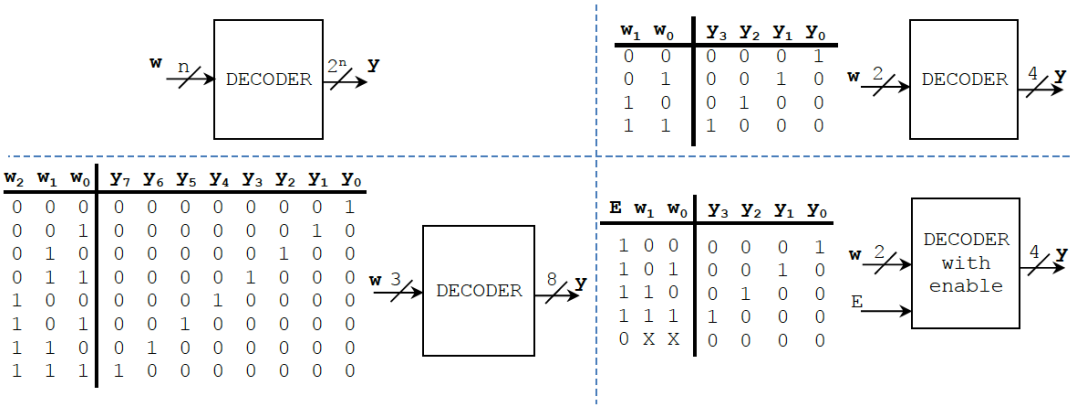
Démultiplexeurs :

- Un démultiplexeur est l'opposé du multiplexeur.
- Ce circuit est utilisé par exemple pour envoyer sur un même canal de l'informations en provenance de plusieurs sources (multiplexage/démultiplexage temporel)



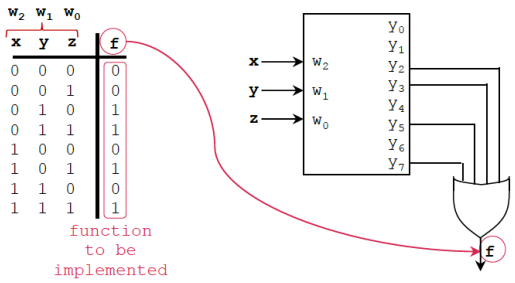
Décodeurs

- Un décodeur a *n* entrées et 2^n sorties.
- De façon général, un décodeur est un circuit qui transforme les entrées en sorties en suivant certaines règles du moment que le nombre d'entrée est supérieur ou égal au nombre de sortie.
- Exemples de décodeurs 2-vers-4, 3-vers-8, 2-vers-4 avec signal d'activation. Le signal y_i est activé lorsque la valeur décimale de l'entrée *w* est égal à *i*.



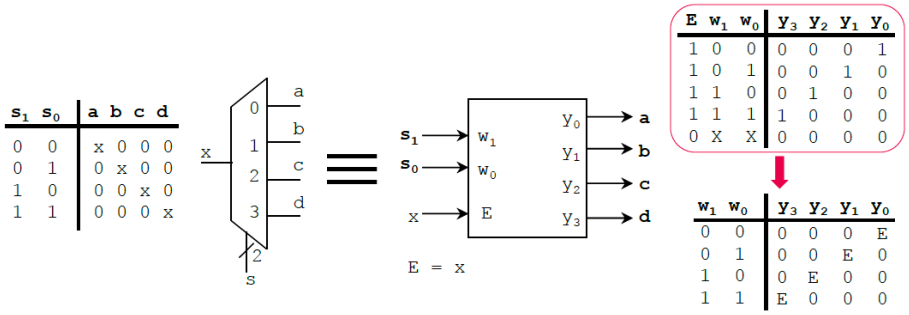
Circuits logiques à base de décodeurs

- Les décodeurs peuvent être utilisés pour implémenter les fonctions booléennes. Remarquez que les sorties correspondent aux minterms.
- Dans l'exemple suivant, le minterm 2 est activé lorsque *xyz=010*, donc y_2 est 1. y_5 et y_7 sont aussi activés lorsque *xyz* prennent pour valeurs 101 et 111 respectivement.



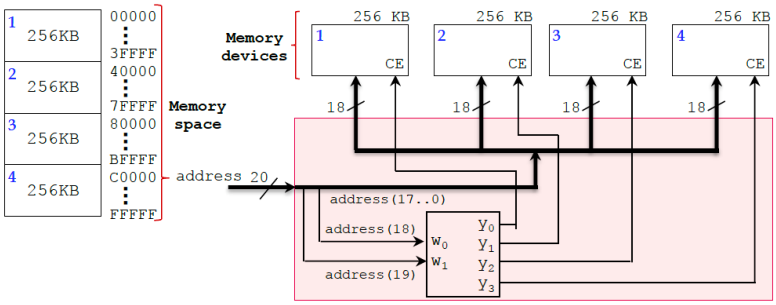
Démultiplexeurs à base de décodeurs

- En utilisant le signal d'activation d'un décodeur comme signal d'entré, il est possible de réaliser un démultiplexeur avec un décodeur.



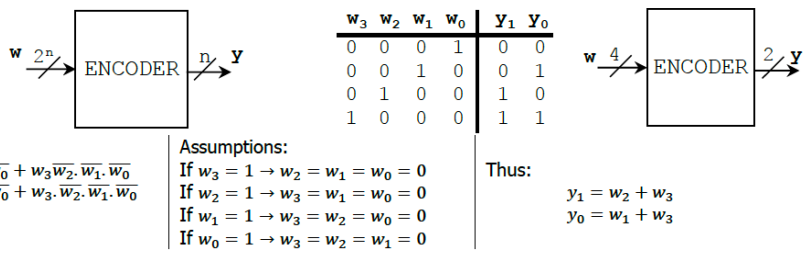
Application : Décodeur mémoire

- Un microprocesseur ayant 20 lignes pour représenter les adresses peut gérer 2^{20}=1 Mo d'adresses, chaque adresse contenant 1 octet d'information. On veut connecter 4 modules mémoires de 256 Ko à ce microprocesseur.
- Le circuit en rouge permet à partir d'une adresse sur 20 bits reçue en entrée de sélectionner le bon module mémoire (parmi les 4) et de l'activer via l'envoie du signal CE (*Chip Enable*). (ex : @=0XD0123, seul le module 4 est activé).



Encodeurs

- Un encodeur a 2^n entrées et n sorties. C'est exactement l'opération opposée du décodeur : lorsque le signal w_i est activé, l'index i apparaît sur la sortie y (en binaire).
- Exemple d'encodeur 4-vers-2



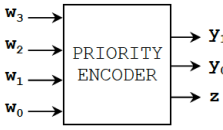
$y_1 = \overline{w_3} w_2 \overline{w_1}.\overline{w_0} + w_3 \overline{w_2}.\overline{w_1}.\overline{w_0}$
$y_0 = \overline{w_3}.\overline{w_2} w_1 \overline{w_0} + w_3.\overline{w_2}.\overline{w_1}.\overline{w_0}$

- Remarques :
  - Si plusieurs entrées sont activées, alors la sortie est indéfinie.
  - Si aucune entrée n'est activée, alors la sortie est indéfinie.

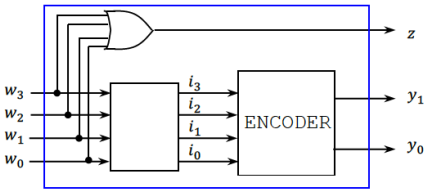
Encodeurs de priorité

- Encodeur standard : On ne fait que tester si une entrée donnée est activée afin que la sortie prenne la bonne valeur.

- Si plusieurs entrées sont activées en même temps, une première solution est d'utiliser une sortie supplémentaire qui est activé pour indiquer que ce comportement est apparu.
- Une solution alternative est d'utiliser un encodeur de priorité. Si plusieurs entrées sont activées, alors on ne considère que l'entrée w_i ayant la valeur de i la plus grande. Si aucune entrée n'est activée, il est alors possible d'ajouter une sortie supplémentaire z positionnée à 0 si aucune entrée n'est activée et 1 sinon.
- En observant la table de vérité de l'encodeur de priorité 4-vers-2 on obtient les fonctions booléennes suivantes :
  - $y_1 = w_2\overline{w_3} + w_3$
  - $y_0 = \overline{w_3}\,\overline{w_2}w_1 + w_3$
  - $z = \overline{w_3}\,\overline{w_2}\,\overline{w_1}\,\overline{w_0} = w_3 + w_2 + w_1 + w_0$
- Il est possible de simplifier y_1 et y_0 (Algébriquement, K-maps, Quine-McCluskey). Une alternative consiste à générer les signaux intermédiaires $i_3 = w_3$; $i_2 = \overline{w_3}w_2$; $i_1 = \overline{w_3}\,\overline{w_2}w_1$; $i_0 = \overline{w_3}\,\overline{w_2}\,\overline{w_1}w_0$ de sorte que ces signaux soit exclusifs et peuvent ainsi être utilisés en entré d'un encodeur classique.

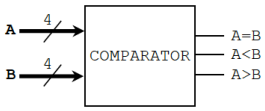


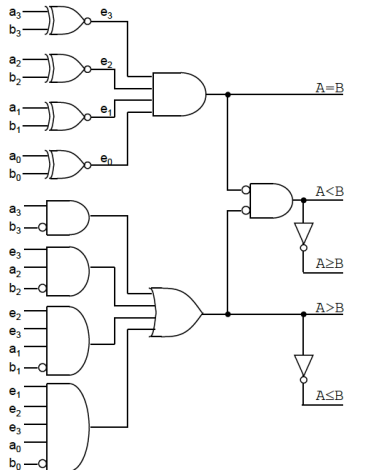
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	0	0	0
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1



Comparateurs

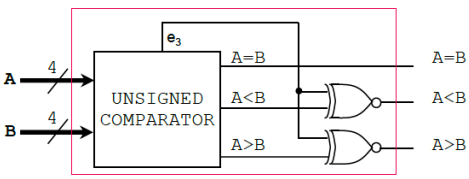
- **Comparateur de nombre non signé :**
- Soit $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$ alors A>B lorsque :
  - $a_3 = 1, b_3 = 0$
  - ou : $a_3 = b_3$ et $a_2 = 1, b_2 = 0$
  - ou : $a_3 = b_3, a_2 = b_2$ et $a_1 = 1, b_1 = 0$
  - ou : $a_3 = b_3, a_2 = b_2, a_1 = b_1$ et $a_0 = 1, b_0 = 0$





- **Comparateur de nombre signé :**
  - Si les 2 nb sont de même signe, on peut utiliser un comparateur non signé.
  - Si les 2 nb sont de signe opposé, on **ne** peut **pas** utiliser un comparateur non signé directement.
  - Pour des nombres en complément à 2 :
    - Si $a_3 \neq b_3$, il faut inverser les bits de A>B et A<B du comparateur non signé.
    - Si $a_3 = b_3$, on ne fait rien.

$$(A < B)_{signed} = \overline{e_3}\oplus(A < B)_{unsigned} = \overline{e_3\oplus(A < B)_{unsigned}}$$
$$(A > B)_{signed} = e_3\oplus(A > B)_{unsigned}$$



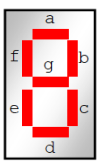
- **Comparateur de nombre signé/non signé :**
  - Une version plus simple consiste à utiliser un additionneur en complément à 2 (A+/-B). Si le résultat est positif (bit le plus à gauche à 0) alors $A \geq B$, sinon A<B.

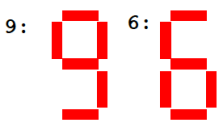
Conversion d'encodage

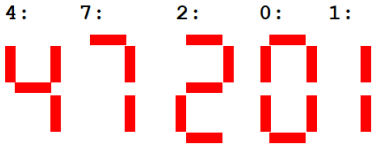
Décodeur BCD vers 7-segment

- Le système BCD (*Binary Coded Decimal*) est utile car il fournit un format lisible par l'humain. Par exemple un clavier numérique fournit un code de 4-bit au format BCD.
- Le convertisseur BCD vers 7-segment est un décodeur car le nombre de sortie est supérieur au nombre d'entrée.

b_3	b_2	b_1	b_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X







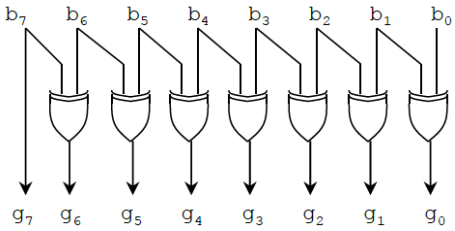
Décodeur Code de Gray vers BCD

- Le convertisseur Code de Gray vers BCD est un décodeur car le nombre de sortie est égal au nombre d'entrée.
- Voici la table de vérité dans le cas 4-bit.

$g_3g_2g_1g_0$	$b_3b_2b_1b_0$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 0
0 0 1 0	0 0 1 1
0 1 1 0	0 1 0 0
0 1 1 1	0 1 0 1
0 1 0 1	0 1 1 0
0 1 0 0	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 0 0 1
1 1 1 1	X X X X
1 1 1 0	X X X X
1 0 1 0	X X X X
1 0 1 1	X X X X
1 0 0 1	X X X X
1 0 0 0	X X X X

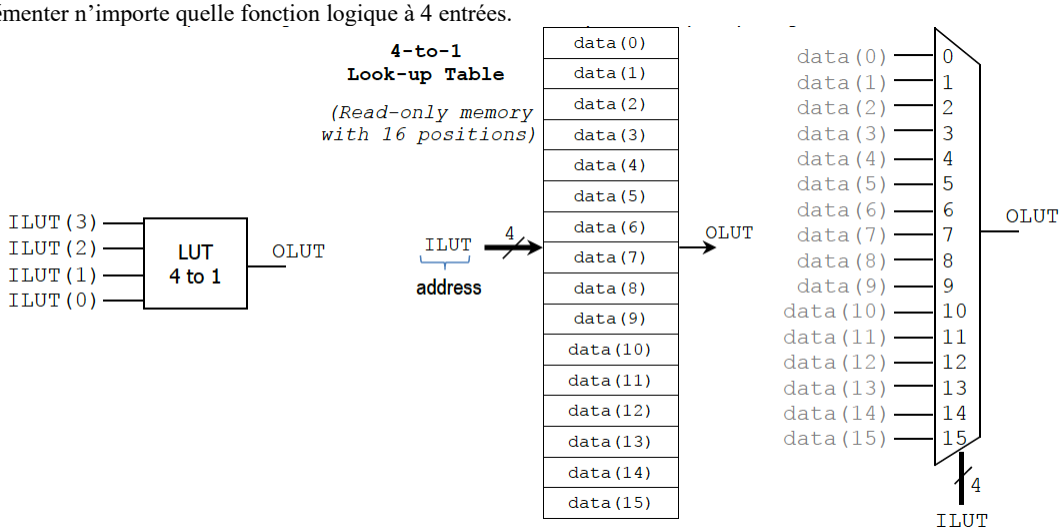
Décodeur Binaire vers Code de Gray

- Le convertisseur Binaire vers Code de Gray est un décodeur car le nombre de sortie est égal au nombre d'entrée.
- Pour un petit nombre on peut utiliser la table de vérité, cependant pour les grands nombres le circuit suivant est plus efficace :



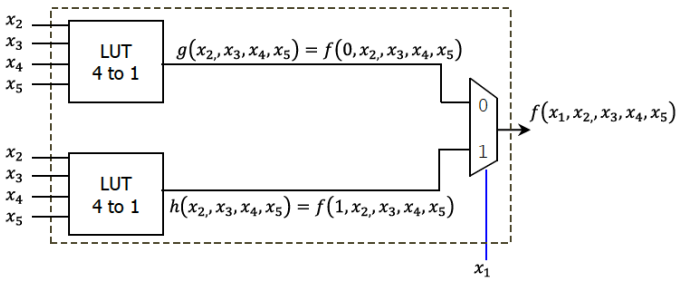
Look-Up Tables (LUT)

- Une LUT 4-vers-1 peut être vue comme une ROM à 16 adresses, chaque adresse contenant 1 bit d'information. Elle peut être vue comme un multiplexeur avec un nombre fixe d'entrée. C'est la façon dont les FPGA implémentent les LUT. Une LUT 4-vers-1 peut implémenter n'importe quelle fonction logique à 4 entrées.



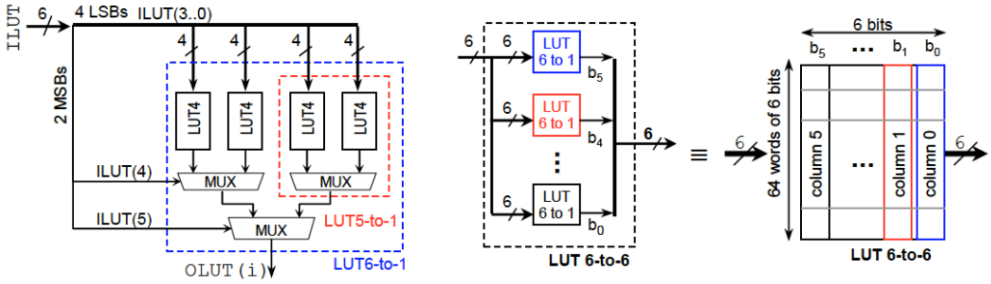
Décomposition de LUT en utilisant l'expansion de Shannon

- L'expansion de Shannon est une méthode permettant d'implanter une fonction booléenne quelconque à l'aide de LUT et de multiplexeur. Une fonction booléenne à *n* variables peut être décomposée en 2 fonctions booléennes à *(n-1)* variables suivies d'un MUX.



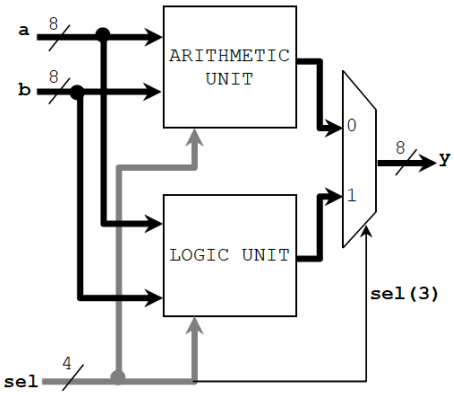
LUT plus grosses

- Il est possible de construire les LUT plus grosses offrant plus de possibilités en combinant les LUT entre elles, suivies d'un multiplexeur. On peut construire ainsi des LUT NI-vers-1. Par exemple, une LUT 6-vers-1 peut être construite à partir de deux LUT 5-vers-1, chaque LUT 5-vers-1 étant elle-même la combinaison de 2 LUT 4-vers-1.



Unité Arithmétique et Logique (UAL)

- Les UAL offrent deux types d'opérations : arithmétique et logique (bit à bit).
- Dans l'exemple suivant, l'entrée *sel(3..0)* sélectionne l'opération, et *sel(2..0)* le type d'opération à l'intérieure de l'unité.



sel	Operation	Function	Unit
0 0 0 0	y <= a	Transfer 'a'	ARITHMETIC
0 0 0 1	y <= a + 1	Increment 'a'	
0 0 1 0	y <= a - 1	Decrement 'a'	
0 0 1 1	y <= b	Transfer 'b'	
0 1 0 0	y <= b + 1	Increment 'b'	
0 1 0 1	y <= b - 1	Decrement 'b'	
0 1 1 0	y <= a + b	Add 'a' and 'b'	
0 1 1 1	y <= a - b	Subtract 'b' from 'a'	
1 0 0 0	y <= NOT a	Complement 'a'	LOGIC
1 0 0 1	y <= NOT b	Complement 'b'	
1 0 1 0	y <= a AND b	AND	
1 0 1 1	y <= a OR b	OR	
1 1 0 0	y <= a NAND b	NAND	
1 1 0 1	y <= a NOR b	NOR	
1 1 1 0	y <= a XOR b	XOR	
1 1 1 1	y <= a XNOR b	XNOR	

Décalage à barillet

- Il y a deux types de décalage :
  - Arithmétique : (*mode=0*) lors du décalage à droite on utilise l'extension de signe qui consiste à recopier le signe de façon à préserver le signe d'un nombre.
  - Logique : (*mode=1*) lors du décalage à droite on insert des 0.
- La table de vérité d'une unité de décalage à 8-bit est la suivante :
  - *result[0..7]* : c'est la sortie décalée (shift) de l'entrée *data[7..0]*
  - *sel[2..0]* : nombre de bits à shifter
  - *dir* : contrôle la direction du décalage (*dir=1* : droite, *dir=0* : gauche)

mode = 0. ARITHMETIC MODE				mode = 1. ROTATION MODE			
dir	**dist[2..0]**	**data[7..0]**	**result[7..0]**	**dir**	**dist[2..0]**	**data[7..0]**	**result[7..0]**
X	0 0 0	abcdefgh	abcdefgh	X	0 0 0	abcdefgh	abcdefgh
0	0 0 1	abcdefgh	bcdefgh0	0	0 0 1	abcdefgh	bcdefgha
0	0 1 0	abcdefgh	cdefgh00	0	0 1 0	abcdefgh	cdefghab
0	0 1 1	abcdefgh	defgh000	0	0 1 1	abcdefgh	defghabc
0	1 0 0	abcdefgh	efgh0000	0	1 0 0	abcdefgh	efghabcd
0	1 0 1	abcdefgh	fgh00000	0	1 0 1	abcdefgh	fghabcde
0	1 1 0	abcdefgh	gh000000	0	1 1 0	abcdefgh	ghabcdef
0	1 1 1	abcdefgh	h0000000	0	1 1 1	abcdefgh	habcdefg
1	0 0 1	abcdefgh	aabcdefg	1	0 0 1	abcdefgh	habcdefg
1	0 1 0	abcdefgh	aaabcdef	1	0 1 0	abcdefgh	ghabcdef
1	0 1 1	abcdefgh	aaaabcde	1	0 1 1	abcdefgh	fghabcde
1	1 0 0	abcdefgh	aaaaabcd	1	1 0 0	abcdefgh	efghabcd
1	1 0 1	abcdefgh	aaaaaabc	1	1 0 1	abcdefgh	defghabc
1	1 1 0	abcdefgh	aaaaaaab	1	1 1 0	abcdefgh	cdefghab
1	1 1 1	abcdefgh	aaaaaaaa	1	1 1 1	abcdefgh	bcdefgha