Systèmes Digitaux

Cours 4

Types de circuits

- Il existe 2 types de circuits:
  - Circuits combinatoires
  - Circuits séquentiels
- Avec les circuits combinatoires, la sortie ne dépend **QUE** de l'entrée
- D'autres ont une sortie qui ont une mémoire:
  - Ils peuvent aussi dépendre des valeurs précédentes
  - On appelle ça des circuits séquentiels

Pour l'instant, on se concentre sur les circuits combinatoires

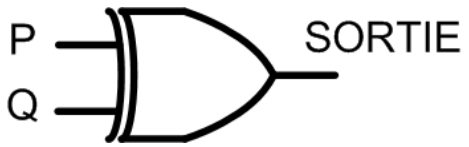
Circuits combinatoires

- Les circuits vus jusqu'à présent sont tous combinatoires
  - Il existe cependant des composantes classiques qui méritent une attention particulière
  - On se concentre sur ces composantes là aujourd'hui
- On va passer au travers de chacun et explorer ce qu'on peut faire avec...
  - Commençons par un petit rappel...

Retour en arrière

- La porte OU-exclusif (XOR) fait le travail suivant:

P	Q	SORTIE
0	0	0
0	1	1
1	0	1
1	1	0



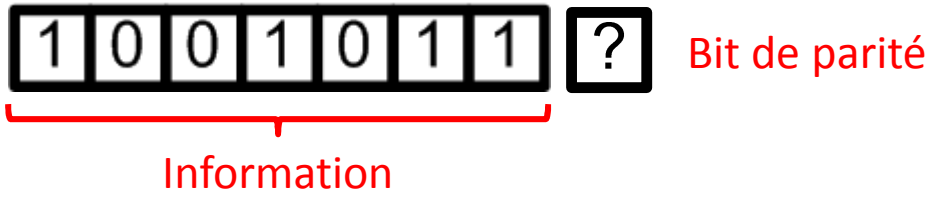
- Il donne ‘1’ quand les entrées sont différentes
- Le XOR est utilise dans plusieurs systèmes importants...

Allons voir quelques exemples d’application...

Détection d'erreur

- Quand on envoie des données, il se peut qu'il y ait erreur
  - S'il y en a, on peut retransmettre

- Comment détecter s'il y a erreur ou pas?
  - Un bit de parité
  - On peut envoyer l'information et ajouter 1 bit
  - Ce bit nous aidera à savoir si la donnée est bonne ou pas



Detection d'erreur

- La parité peut etre paire ou impaire:
  - Ça veut dire que le nombre de '1' sera pair/impair
- Si on choisissait "pair"...
  - On envoie l'information et on compte le nombre de '1'
  - Si le nombre de '1' est pair, on ajoute un '0'
  - Si le nombre de '1' est impair, on ajoute '1' pour que ça devienne pair
- Ex: on veut envoyer la séquence 1001011
  - Avec parité paire: 10010110 et impaire 10010111

Additionneur

- On se sert aussi des XOR dans les additionneurs (et soustracteurs):
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=0 (avec retenue de 1)
- Pour additionner 2 termes de 1 bit, j'ai besoin d'un XOR
  - À la sortie, j'ai 1 somme et 1 retenue

On appelle ça un demi-additionneur

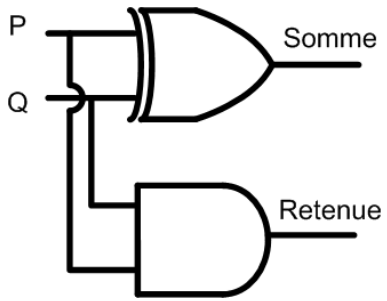
Additionneur

- On peut écrire le tableau de vérité de l'additionneur:

P	Q	Somme	Retenue
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

C'est ce qu'on appelle un demi-additionneur

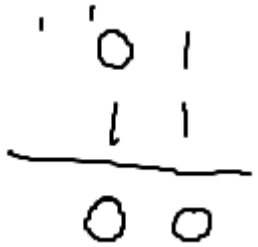
- La somme est un XOR et la retenue est un ET:



Alors, on sait comment additionner P et Q de 1 bit

Additionneur

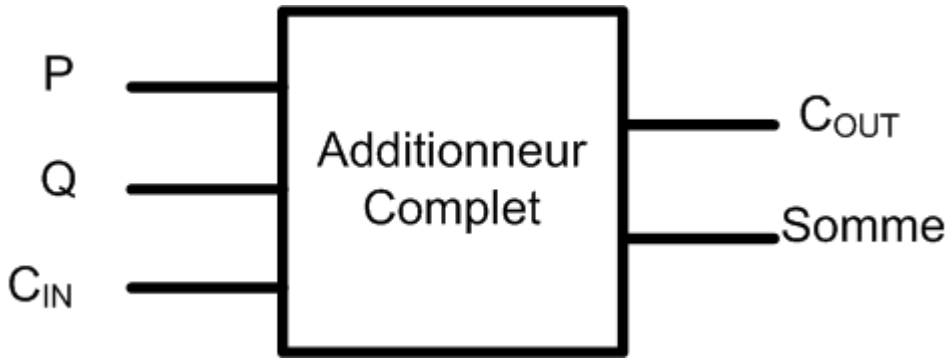
- Comment fait-on pour additionner 2 bits?



  - On additionne les premiers bits: on génère une somme
  - Une retenue peut être générée
  - On additionne les deuxièmes bits avec la retenue du premier bit
  - On doit donc additionner 3 termes: 1 bit de chaque terme et la retenue du bit précédent

Additionneur

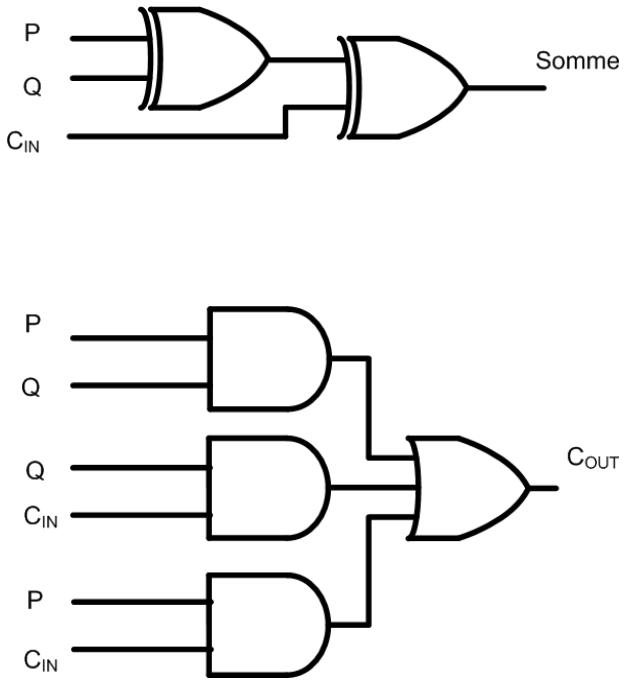
- Notre additionneur devrait avoir 3 entrées:
  - 1 bit pour P, 1 bit pour Q et une retenue en entrée
- Notre additionneur devrait avoir 2 sorties
  - Somme et retenue



P	Q	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Additionneur

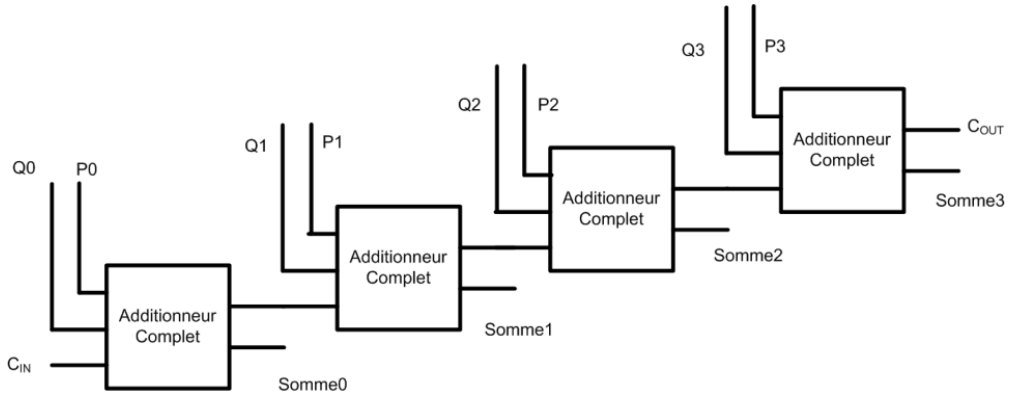
- En analysant plus en détails, on serait capable de dessiner un circuit:



P	Q	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

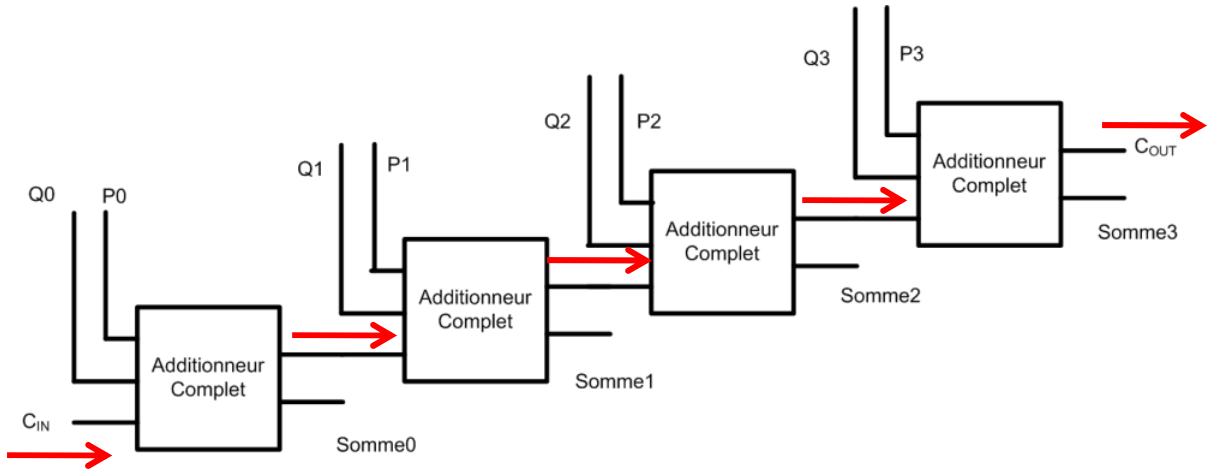
Additionneur

- Pour les additionneurs à N bits, c'est pareil:
  - La sortie du 2e bit génère aussi une retenue qui est transmise au 3e bit
  - La sortie du 3e bit génère aussi une retenue qui est transmise au 4e bit
  - Le seul bit qui n'a pas de retenue en entrée c'est le premier



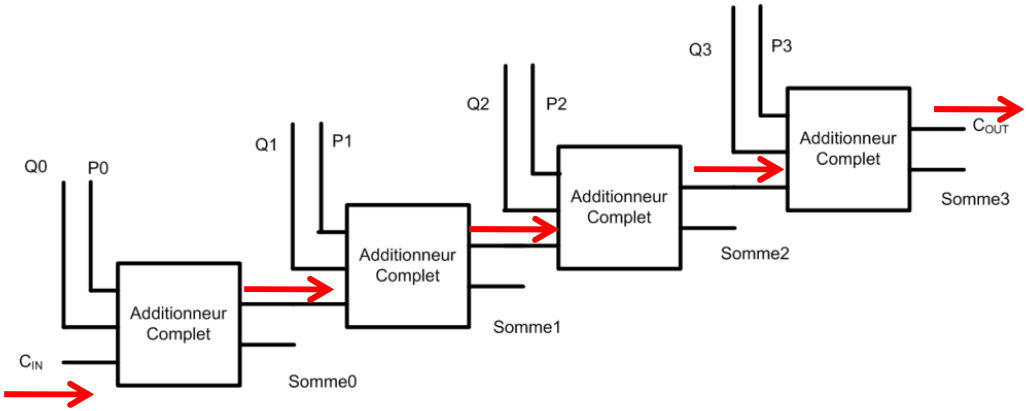
Propagation de retenue

- Ça nous donne la configuration d'additionneur de base
  - Additionneur à propagation de retenue ("Carry-ripple adder")
  - On voit que la retenue se propage d'un étage à l'autre
  - Pour avoir plus de bits, on ajoute d'autres étages...



Propagation de retenue

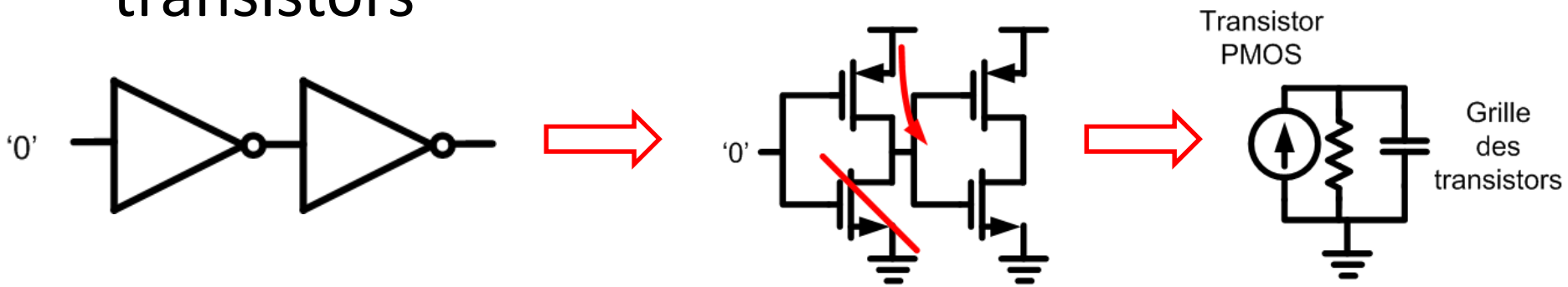
- L'additionneur à propagation de retenue a 2 grandes caractéristiques:
  - Petite taille
  - Lent
- On choisit ce genre de circuit quand la vitesse n'est pas très importante



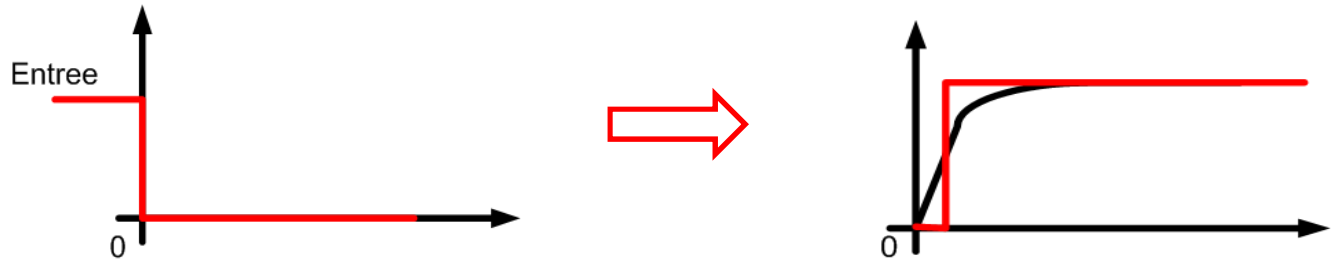
Pourquoi dit-on que c'est lent?

Délai

- Rappel: Les portes logiques sont faites de transistors



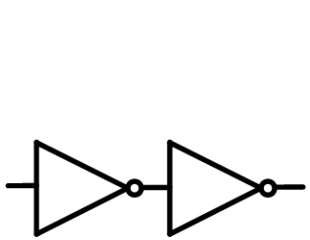
  - On regarde la sortie du premier inverseur
  - Il y a un certain délai entre l'entrée et la sortie



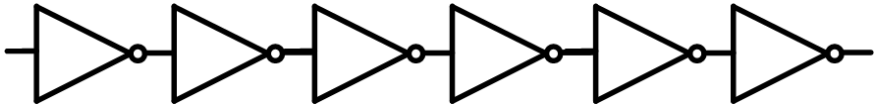
Il y a donc toujours un délai entre l'entrée et la sortie d'une porte logique...

Délai

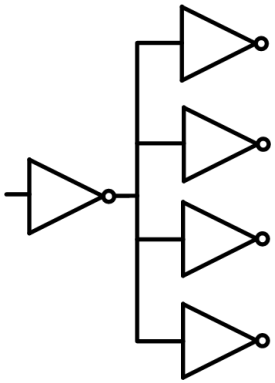
- Ce délai dépend de:
  - La taille de la porte: il existe des portes qui donnent plus de courant (plus rapides)
  - La capacité à commander: plus de grille de transistors, plus de capacité, moins c'est rapide
  - Évidemment: le nombre de portes à "traverser"



Petit délai



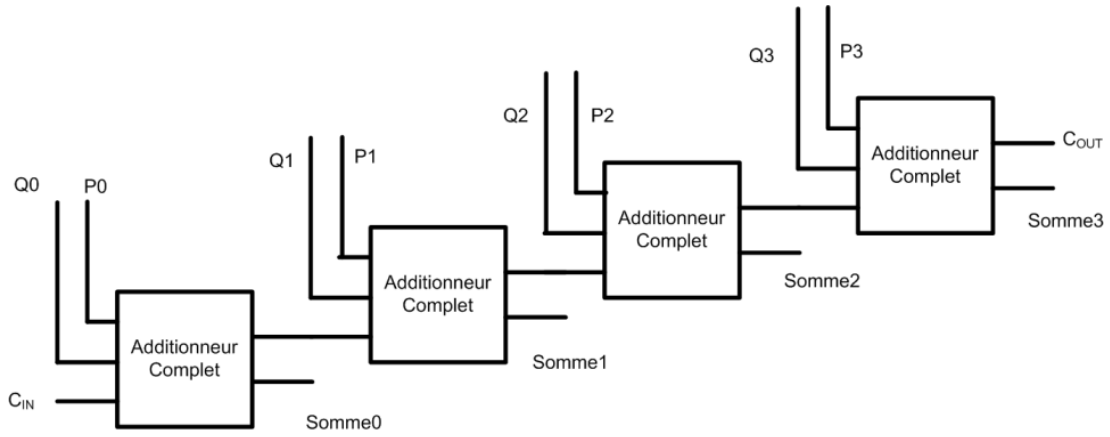
Plus gros délai



Plus gros délai

Délai

- On retourne à notre additionneur de 4 bits



  - Pour avoir *somme1*, il faut qu'on attende que C_{OUT} du premier additionneur arrive.
  - Pour avoir *somme3*, il faut qu'on attende que C_{OUT} du 3^{e}, qui doit attendre le 2^{e} qui doit attendre le 1^{er}...

Si j'avais 16 bits à additionner, le délai peut devenir long...

Délai

- L'additionneur est un des éléments importants
  - On se sert de ça partout dans les microcontrôleurs et pour le traitement de signaux
- Les chercheurs ont essayé de l'améliorer...
  - "Comment réduire le temps de propagation?"
- Il existe plusieurs techniques:
  - Carry look-ahead
  - Carry-save
  - ...

Sachez simplement que c'est un enjeu important...

Soustracteur

- Il existe 2 grandes familles de soustracteurs:
  - La version classique
  - La version en complément à 2
- Dans la version classique, on procède comme avec l'additionneur:
  - On fait la table de vérité: A-B

A	B	Difference	Emprunt
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Ça fait un demi-soustracteur

Soustracteur

- Pour le rendre plus complet, on doit ajouter une 3^{e} entrée:
  - L'emprunt de l'étage précédente

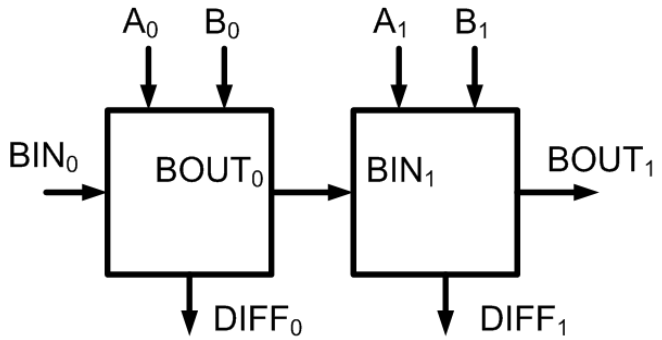
BIN	A	B	Difference	BOUT
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

C'est le soustracteur complet

On peut faire son circuit avec Karnaugh si on le voulait...

Soustracteur

- On peut connecter des soustracteurs complets ensemble comme avec les additionneurs



- Et c'est la chaîne de retenue (d'emprunt) qui risque de prendre beaucoup de temps...

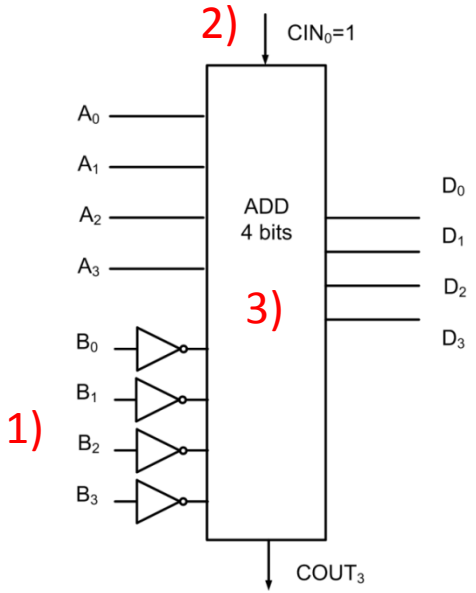
Parlons maintenant de l'autre méthode...

Soustracteur

- L'autre option c'est le complément à 2
- Imaginons qu'on veuille faire A-B:
  - On pourrait prendre –B et l'additionner à A
  - -B devrait être en complément à 2
- Comment mettre –B en complément à 2?
  - On inverse les bits
  - Et on additionne 1
- Ça s'implante bien avec un additionneur...

Soustracteur

- L'idée est donc d'additionner A et (–B)
  1) Pour avoir –B, on fait le complément à 1 (inversion)
  2) On additionne 1 au complément à 1 pour avoir un complément a 2 (CIN_0=1)
  3) Et on additionne normalement...



Overflow

- Quand on additionne/soustrait 2 nombres il peut arriver 1 de 2 choses:
  - Tout peut fonctionner normalement
  - OU, le résultat final pourrait ne pas être bon
- Ceci est dû à un phénomene appelé “overflow”
  - En français: “Débordement”
  - Si la réponse prend plus d’espace que ce qui lui est alloué, il peut y avoir overflow

Overflow

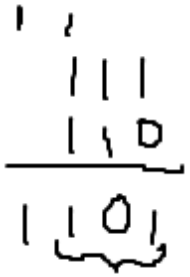
- Ex: En additionnant 2 nombres de 2 bits non signés:
  - $(01)_2+(01)_2=(10)_2$
  - Le resultat est 2, ce qui est bon
- Ex: En additionnant 2 nombres de 2 bits non signés
  - $(11)_2+ (11)_2=(110)_2$ (mais j'ai droit a 2 bits!)
  - Le résultat me donnerait $(10)_2$ qui n'est pas la bonne réponse: on dira qu'il y a "overflow"

Overflow

- Dans un nombre non-signé, un “overflow” se produit quand le résultat est trop gros:
  - S’il y a une retenue au DERNIER bit (plus significatif), il y a overflow
  - Sinon, il n’y en a pas...
- Pour les nombres signés, il faut y réfléchir un peu plus...

Overflow

- Prenons un cas avec 3 bits... -1 + -2
  - J'additionne en complément à 2
  - Donc $(111)_2 + (110)_2$

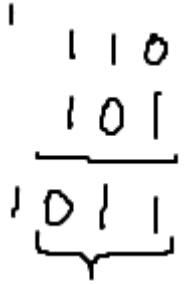


  - On convertit et ça donne -3... c'est bon
  - Même s'il y a une retenue au dernier bit, la réponse est bonne!

Cet indicateur n'est peut-être pas bon

Overflow

- Prenons un autre -2 + -3
  - L'addition devrait me donner -5
  - On additionne en complément à 2:



  - Il y a une retenue...
  - Mais je retrouve 3 à la place de -5.

La retenue au dernier bit ne semble pas être très bon...

Overflow

- En essayant beaucoup d'exemples, on viendrait à la conclusion suivante:
  - Pour ne PAS avoir de overflow, il faut que les 2 **DERNIÈRES** retenues soient égales
  - Donc, si la dernière retenue est 0, celle d'avant devrait être 0
  - Si la dernière retenue est 1, celle d'avant devrait être 1
- Si les 2 dernières retenues sont différentes, il y aura overflow

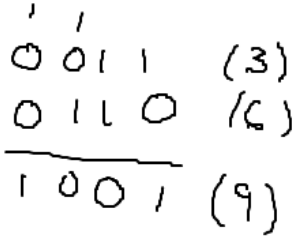
Addition de BCD

- On utilise parfois les nombres binaires pour exprimer des nombres décimales:
  - Chaque décimale peut aller de 0 a 9 (4 bits)
  - Avec 4 bits, il y a 15 possibilités... 6 de ces valeurs ne seront pas utilisées (“on ne les comprend pas”)
  - On appelle ca BCD (“binary coded decimals”)
- Pour exprimer des nombres a 2 chiffres (décimales) on utilise 8 bits:
  - Par exemple: $(76)_{10}$ serait $(0111\ 0110)_2$

7 6

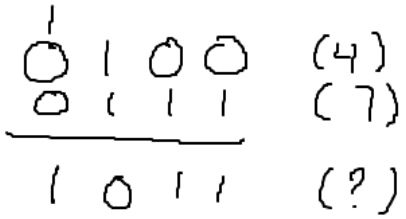
Addition de BCD

- Comment fait-on pour additionner 2 BCD?
- Prenons l'exemple de 3+6:



L'addition normale fonctionne

- Prenons maintenant 4+7:

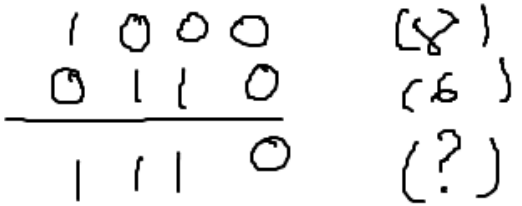


En BCD, 1011 ne représente rien

L'addition normale ne fonctionne pas toujours pour BCD

Addition de BCD

- Mais qu'est-ce que ça devrait donner en BCD?
  - En décimale, 4+7 devrait donner $(11)_{10}$
  - Donc: $(0001\ 0001)_2$ à la place de $(1011)_2$
- En regardant les chiffres, on peut voir qu'il y a une différence de 6 entre 0001 0001 et 1011
- Faisons un autre test:
  - J'additionne 8+6...



Ça aurait du être 0001 0100

Addition de BCD

- La solution semble être d'additionner 6 au résultat final
  - Mais pas en tout temps!
  - Seulement quand la somme est plus que 9
- Il faut donc:
  1) Additionner les termes normalement
  2) Voir si c'est plus grand que 9
  3) Si oui, il faut ajouter 6 a la réponse
  4) Sinon, on ne fait rien..

Addition de BCD

- Comment faire pour détecter si c'est plus que 9?

S_3S_2 \ S_1S_0	00	01	11	10
00				
01				
11	1	1	1	1
10			1	1

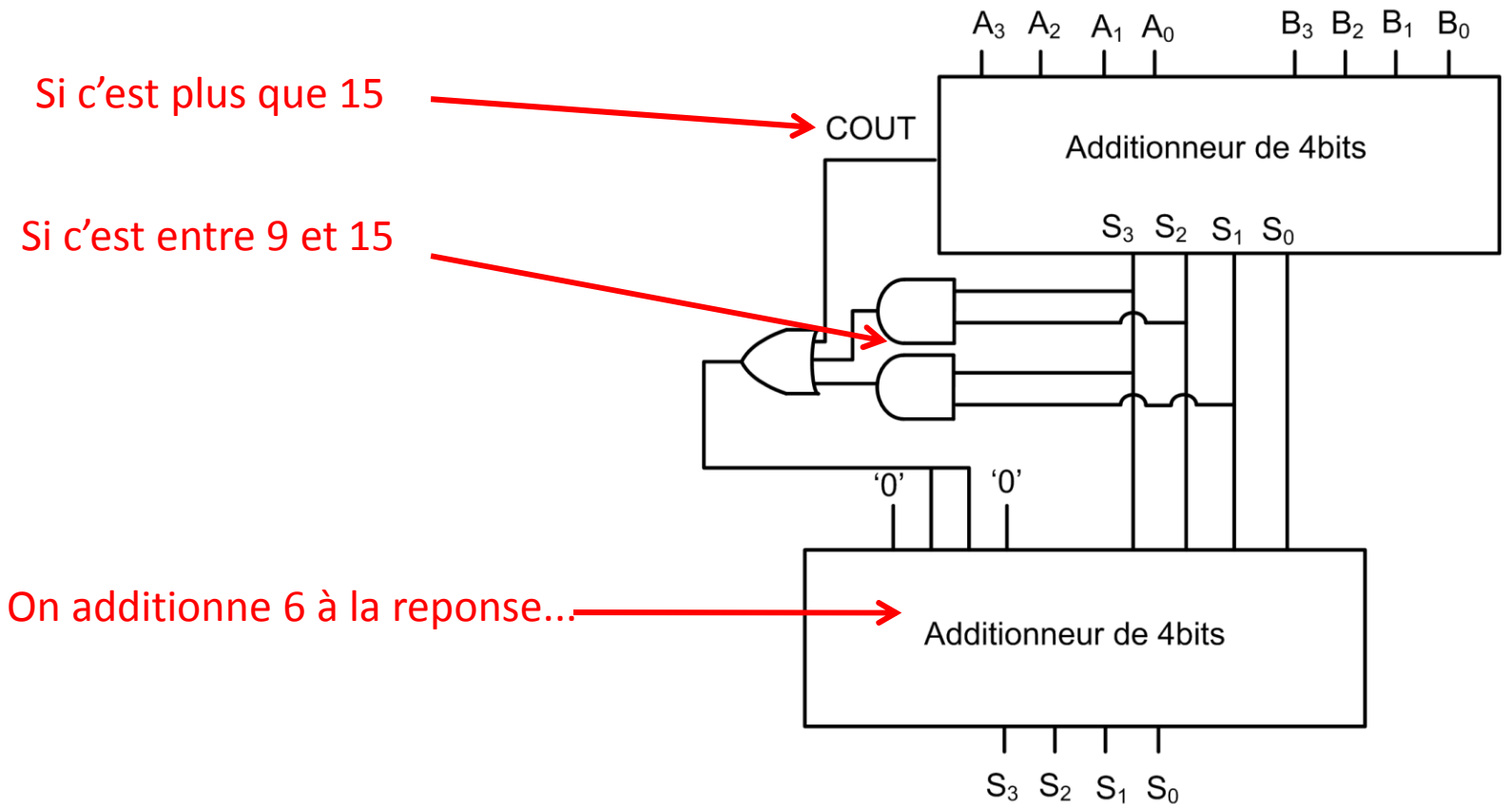
$$S_3S_2 + S_3S_1$$

- Est-ce que c'est tout?
  - Qu'arrive-t-il quand on additionne 9+9?
  - Le résultat est 18 et ce n'est pas dans la table de Karnaugh

Heureusement , quand c'est plus que 15, l'additionneur donne COUT=1

Addition de BCD

- Diagramme inspire du livre (p.141)



Multiplicateur

- La multiplication est une opération qui demande du temps et des ressources
- C’est habituellement une séquence d’additions
- Quand l’une des entrées est une puissance de 2, ça peut faciliter le travail de beaucoup
- Allons voir pourquoi

Multiplicateur

- Prenons le chiffre 3 en utilisant 6 bits:
  - 000011
- Si je multipliais par 2, j'aurais 6:
  - 000110
- Si je multipliais par 4, j'aurais 12:
  - 001100
- Si je multipliais par 8, j'aurais 24:
  - 011000

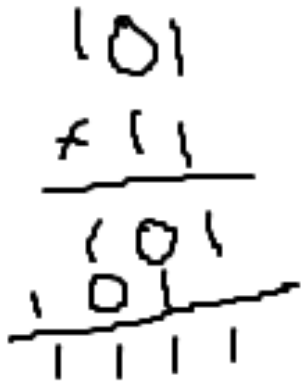
Voit-on quelque chose de particulier?

Multiplicateur

- Une multiplication par une puissance de 2^n implique un décalage de n vers la gauche
  - Une multiplication de 2 implique un décalage de 1
  - Une multiplication de 4 implique un décalage de 2
  - Une multiplication de 8 implique un décalage de 3
  - …
- Si ce n'est pas une puissance de 2, les choses se compliquent.

Multiplicateur

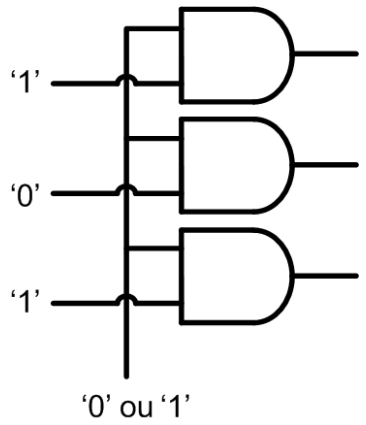
- Faisons une multiplication en binaire pour voir la procédure
- On multiplie $(101)_2$ par $(11)_2$:
  - On considère le terme du bas à droite
  - On le multiplie par chaque terme du haut
  - On passe au prochain terme à sa gauche
  - On le multiplie par chaque terme du haut MAIS la sortie est decalée de 1
  - On passe au prochain...



Voyons comment implémenter ce genre de choses...

Multiplicateur

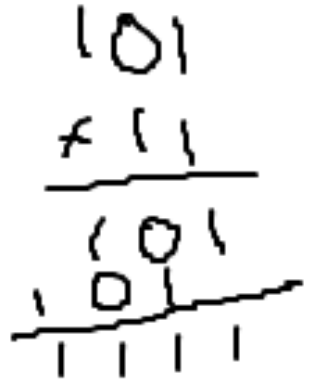
- Essayons de multiplier 101 par 1 bit:
  - Si le bit est 0, la sortie est 000
  - Si le bit est 1, la sortie est 101
- Une façon simple de faire ce circuit c'est avec une porte ET



Essayons ça avec 2 bits...

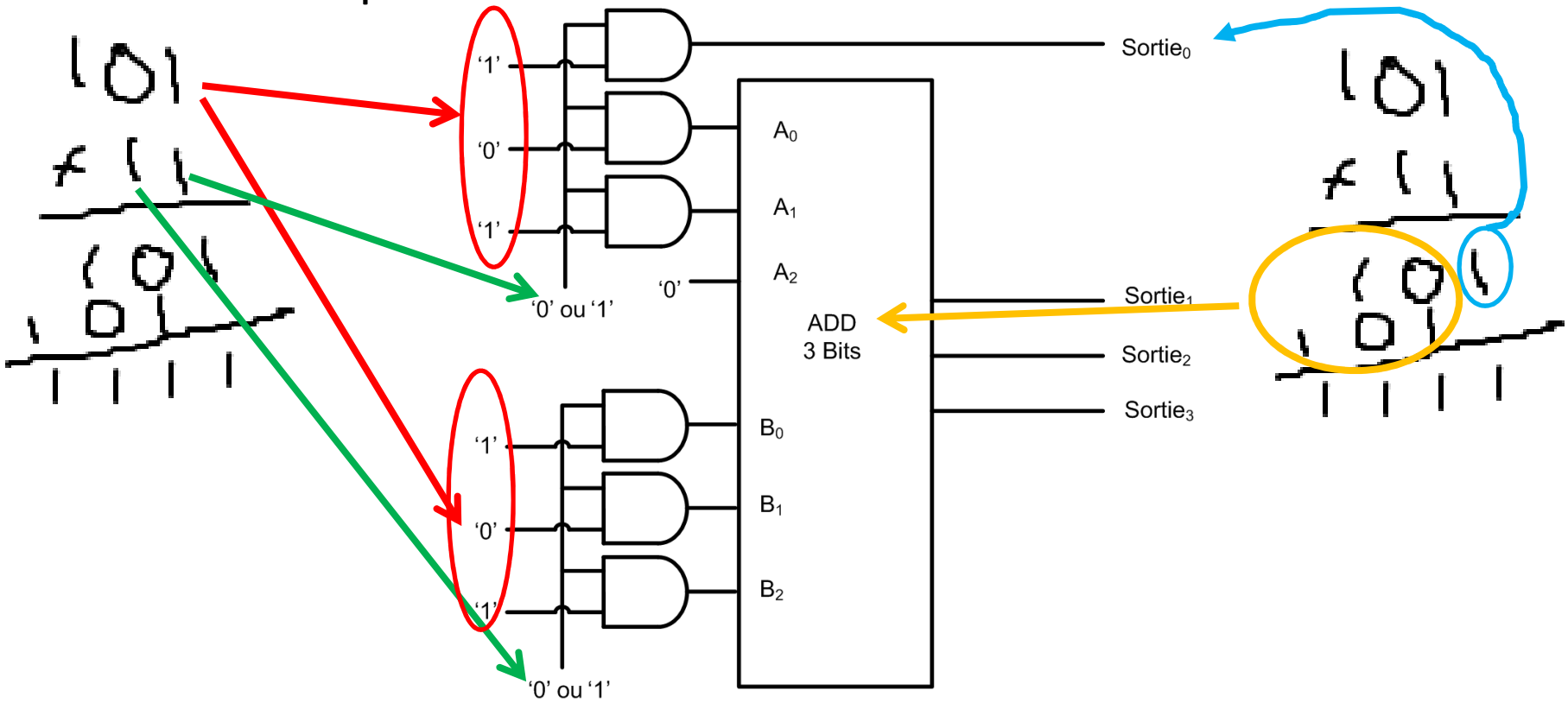
Multiplicateur

- Essayons de multiplier 101 par 2 bits:
  - Si le bit à droite est ‘0’, ça donne 000
  - Si le bit à droite est ‘1’, ça donne 101
- Par la suite, on décale la sortie vers la gauche et on considère le 2^{e} bit:
  - Si le bit à droite est ‘0’, ça donne 000
  - Si le bit à droite est ‘1’, ça donne 101
- Finalement, on additionne



Multiplicateur

- Voici un schéma du circuit:
  - L'équation est copiée 2 fois pour que ce soit plus compréhensible



Comparateur

- Dans plusieurs genres de systèmes, on aura besoin de comparer 2 valeurs:
  - Pensez à une montre electronique: les secondes doivent compter jusqu'à 60 avant de recommencer
  - Comment déterminer si une valeur est plus petite, égale ou plus grande qu'une certaine valeur?
- On utilise un comparateur

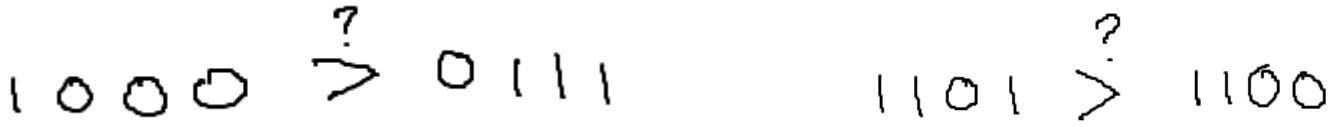
Comparateur

- Il est possible de faire ça avec une table de Karnaugh:
  - On veut comparer 2 nombres de 2 bit: 4 éléments
  - Pour comparer 2 nombres de 3 bits: 6 éléments... c'est déjà plus gros que ce qu'on est capable de faire!
- Heureusement, il y a d'autres façons de faire:
  1) Soustraction et voir si la différence est plus petite, égale ou plus grande que 0
  2) Utiliser une autre structure

Explorons l'option #2

Comparateur

- La méthode de raisonner c'est d'y aller avec une certaine méthodologie:
  - On regarde le bit qui représente le plus gros chiffre ("bit le plus significatif" ou MSB: most significant bit)
  - Si l'un a '1' et l'autre a '0', celui avec '1' est plus gros
  - S'ils sont égaux, on passe au prochain
  - Si l'un a '1' et l'autre a '0', celui avec '1' est plus gros
  - S'ils sont égaux, on passe au prochain...



On traduit en termes plus spécifiques...

Comparateur

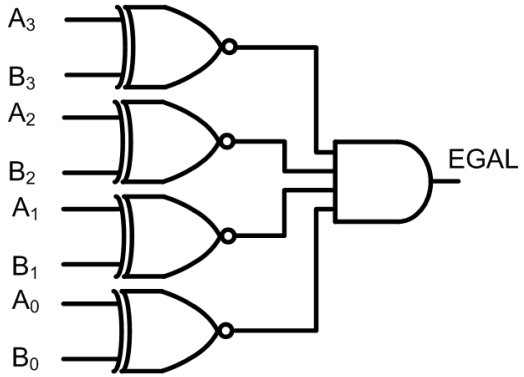
- On compare 2 nombres de 4 bits: A et B
  - A est compose de $A_3A_2A_1A_0$
  - B est compose de $B_3B_2B_1B_0$
- On va commencer par comparer A_3 et B_3
  - S'ils sont egaux, on passe a A_2 et B_2
  - S'ils sont egaux, on passe a A_1 et B_1
  - S'ils sont egaux, on passe a A_0 et B_0

Comparateur

- Pour voir s'ils sont égaux, on peut utiliser le NON-OU-Exclusif (XNOR):

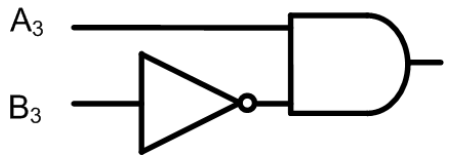
A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

  - Pour A=B, il faut que tous les bits soient égaux
  - A_3 doit être égal a B_3 ET A_2 doit être égal à B_2 ET...

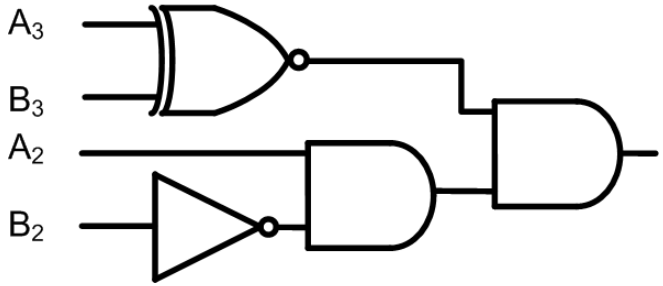


Comparateur

- Pour voir si A > B, on peut utiliser un raisonnement semblable:
  - Si A_3=1 et B_3=0, A est plus grand
  - Et s'ils sont égaux, on passe au prochain bit
  - On continue jusqu'à la fin



A_3=1 et B_3=0

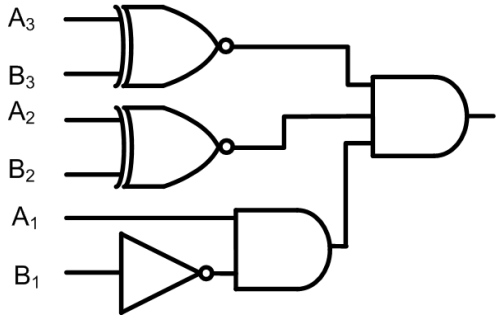


A_3=B_3=0 et A_2 > B_2

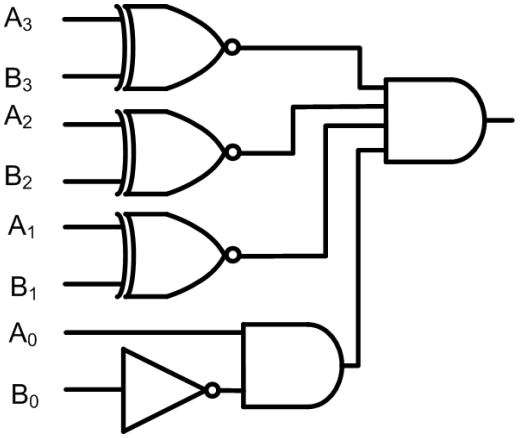
Comparateur

- Si les deux sont égaux, les 2 premiers bits sont égaux, on doit aller voir le 3e:
  - Et finalement, on va voir le 4e bit si les 3 autres sont égaux.
  - À la fin, toutes les parties seront connectées par un OU
  - Il suffit que l'une de ces conditions soit vraie pour que A>B

$A_3=B_3=0$, $A_2=B_2=0$ et $A_1 > B_1$



$A_3=B_3=0$, $A_2=B_2=0$, $A_1=B_1$ et $A_0 > B_0$

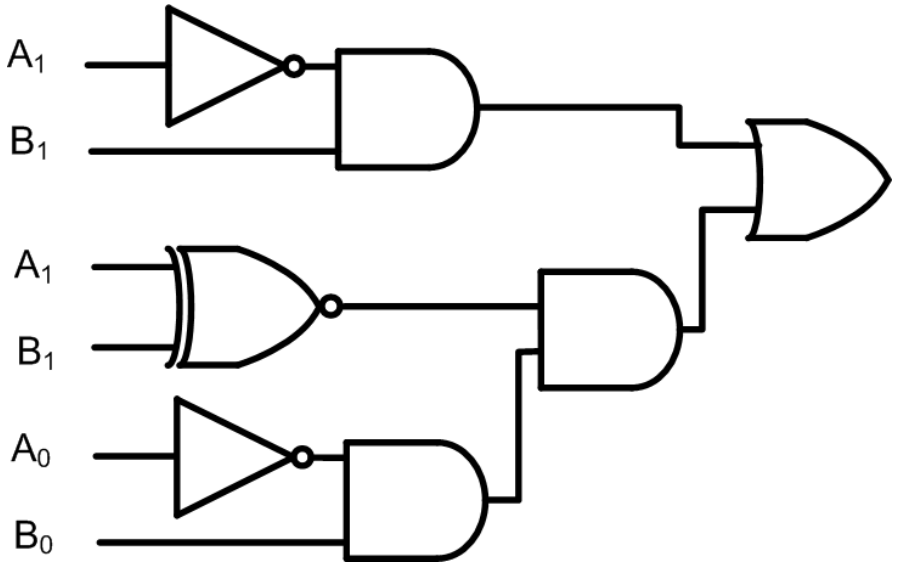


Exemple (seul)

- Refaites la même analyse pour un comparateur à 2 bits pour voir si A < B

Exemple (seul)

- Pour que A < B, il faut que B_1=1 et A_1=0
- S'ils sont égaux, il faut voir le prochain bit:
  - Si B_0=1 et A_0=0, il sera plus grand
  - Rappel: mais ça c'est seulement si A_1=B_1...

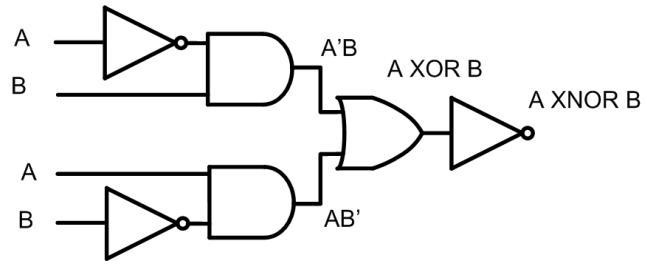


Comparateur

- Chose intéressante (à revoir seul quand vous aurez du temps):
  - Un XNOR est un XOR inverse
  - Un XOR est defini par A'B+AB'
  - Dans A > B on a besoin de AB'
  - Dans A < B on a besoin de A'B
  - Et dans les 3 cas, on a besoin de XNOR
- Si on construisait notre XNOR avec des ET/OU, on pourrait recycler plusieurs parties...

Comparateur

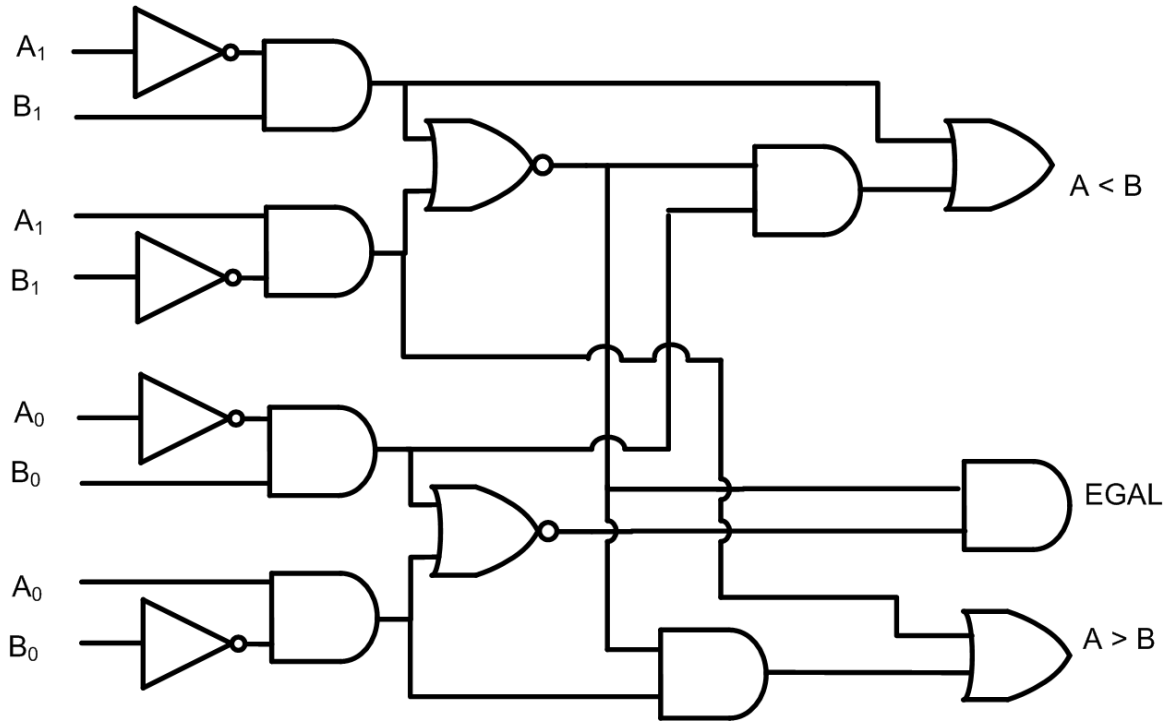
- À la place d'utiliser une porte XNOR toute faite, on peut utiliser ceci:



  - On peut accéder a A'B pour A < B et à AB' pour A > B
  - En plus on peut aussi accéder à A XNOR B
  - Ça réduit le nombre de composantes quand on veut un comparateur qui fait A=B, A>B et A<B en même temps

Comparateur

- Pour un comparateur de 2 bits, ça donne ceci:



Décodeur

- Imaginons un garde de securité qui contrôle l'accès à 4 portes:
  - Les portes mènent à une section différente d'un hotel
  - Si le signal de contrôle est '0' c'est verrouillé et '1' c'est déverrouillé
  - Pour contrôler 4 portes, il a besoin de 4 signaux (donc, 4 boutons)
- Si on le mettait dans un hôtel avec 64 portes, il aurait besoin de 64 boutons...

Décodeur

- À la place de faire ça, on pourrait assigner un numéro binaire à chaque porte:
  - 00 est la porte 1
  - 01 est la porte 2
  - 10 est la porte 3
  - 11 est la porte 4
- On aurait besoin de 2 boutons à la place de 4
  - Avec l'autre exemple de 64 portes, on n'aurait besoin que de 6 boutons ($64=2^6$)

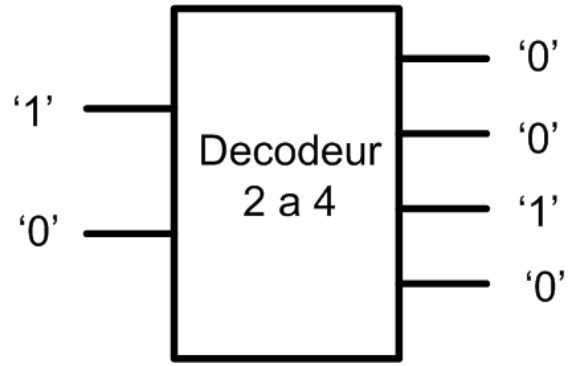
Décodeur

- La question est donc celle-ci
  - Comment transformer 00,01,10,11 en 0001, 0010, 0100, 1000 pour contrôler les portes?
  - En termes techniques: “comment transformer un nombre binaire en one-hot?”
- Réponse: en utilisant un décodeur
  - La fonction du décodeur est de transformer un nombre binaire en one-hot

“one-hot”: nombre binaire contenant des ‘0’ partout sauf à UNE seule place...

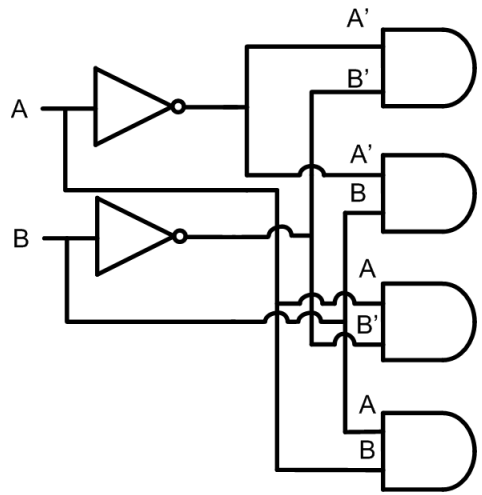
Décodeur

- Si le décodeur avait N entrees, il aura toujours 2^N sorties
  - L'entrée est un nombre binaire
  - La sortie aura toujours UN SEUL '1' est les autres seront des '0'
  - Le '1' sera à la position qui correspond à l'entrée



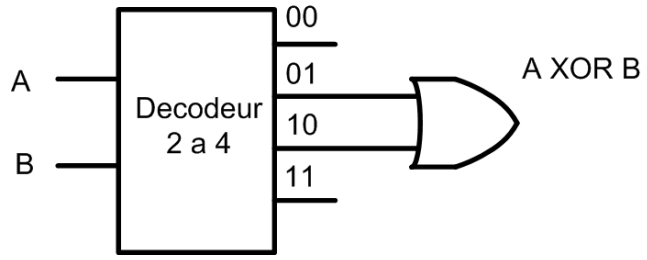
Décodeur

- On peut implanter un décodeur avec la logique suivante:
  - La 1re sortie sera ‘1’ quand l’entrée est 00
  - La 2e sortie sera ‘1’ quand l’entrée est 01
  - La 3e sortie sera ‘1’ quand l’entrée est 10
  - La 4e sortie sera ‘1’ quand l’entrée est 11



Fonctions logiques avec décodeur

- Le décodeur offre une facon facile d'implanter des circuits
  - Tous les minterms sont disponibles
  - On n'a qu'à prendre ceux qui nous intéressent et faire un OU logique
- Pour faire un XOR, par exemple, on s'intéresse aux minterms A'B et AB'...
  - On les connecte à un OU et on a terminé...



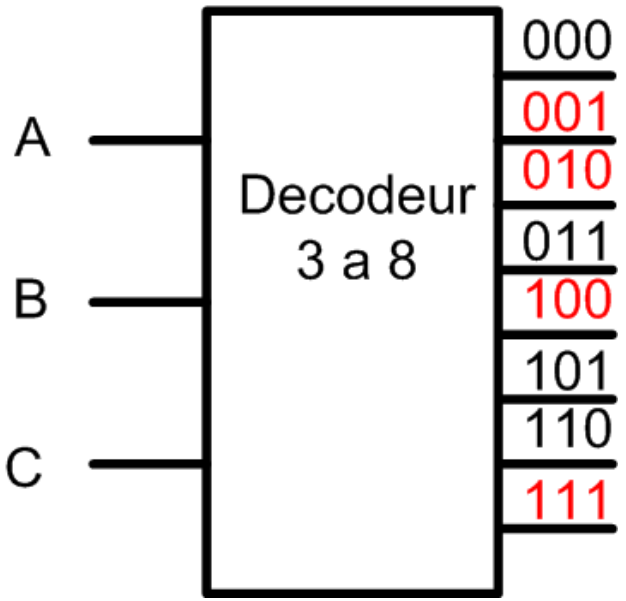
Exemple (seul)

- Implantez la fonction SOMME d'un additionneur avec un décodeur 3 a 8
  - Vous avez le droit d'utiliser un décodeur et une porte OU

Exemple (seul)

- On commence avec la table de verité
- On dessine le décodeur avec les sorties

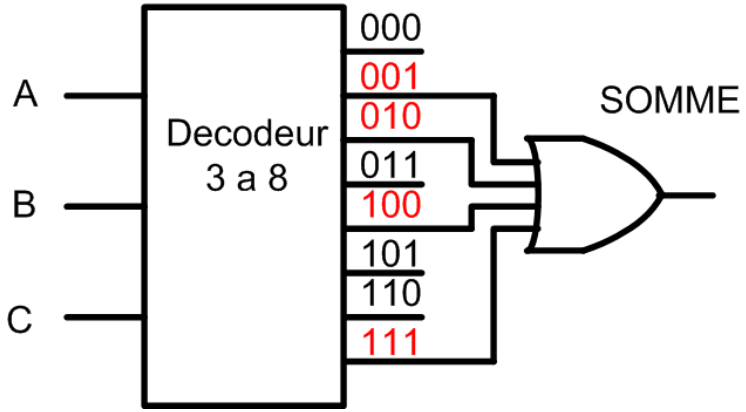
A	B	C	SOMME
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Ce sont les sorties qu'on doit faire entrer dans le OU

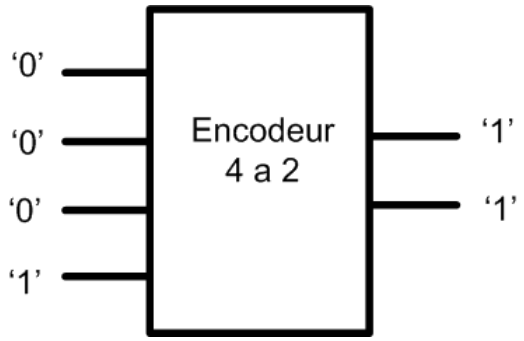
Exemple (seul)

- Le circuit final ressemble donc à:



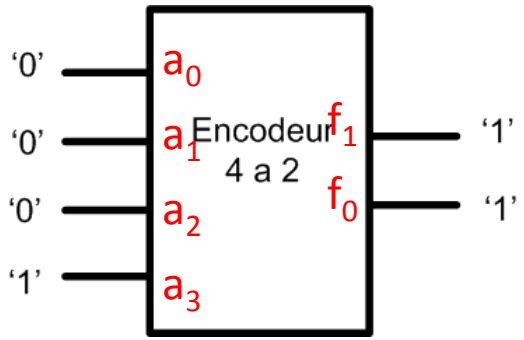
Encodeur

- L'encodeur fait l'opération inverse du décodeur:
  - Il prend un code "one-hot" et le transforme en code binaire
  - Donc, l'entrée a une longueur de 2^N et la sortie a une longueur de N



Encodeur

- On peut determiner son circuit comme ceci:
  - Considérons l'ordre "du haut vers le bas": $A_0A_1A_2A_3$
  - Quand l'entrée est 1000, la sortie est 00
  - Quand l'entrée est 0100, la sortie est 01
  - Quand l'entrée est 0010, la sortie est 10
  - Quand l'entrée est 0001, la sortie est 11



Encodeur

- Avec cette information, on peut dire que
  - Le MSB qui est en position du haut est:

$$H_{MSB} = A_2 + A_3$$

  - Le LSB qui est en position du bas est:

$$B_{LSB} = A_1 + A_3$$

- Ce genre de système fontionne bien quand les règles sont respectéess:
  - Les entrées doivent être en “one-hot”
  - Donc un seul ‘1’ et le reste est ‘0’

Ce n’est pas toujours le cas...

Encodeur de priorité

- Imaginez un systeme de communication de 4 stations
  - Service d'urgence (1000)
  - Sécurité (0100)
  - Entretien (0010)
  - Cuisine (0001)

$a_0a_1a_2a_3$

- Lors d'un appel, le systeme génère un nombre binaire pour identifier la source de l'appel
  - Plusieurs appels peuvent arriver en même temps

Ex: Si la sécurite appelle, le nombre binaire "01" apparaît

Encodeur de priorité

- Si la sécurité et l'entretien appellaient en même temps:

$$H_{MSB} = A_2 + A_3 \qquad B_{LSB} = A_1 + A_3$$

  - Le MSB et le LSB seraient 1: "11"
  - Ça nous afficherait la cuisine!

A_0: Urgence
A_1: Securité
A_2: Entretien
A_3: Cuisine

- Il y a donc un problème si ça fonctionnait comme ça...
  - Le problème est l'hypothèse que les entrées seront toutes "one-hot"

Encodeur de priorité

- Pour résoudre le problème, on utilise un encodeur de priorité:
    - On ne fait plus semblant que c'est du one-hot
- On dit que ça peut être n'importe quoi, MAIS il y a une priorité à respecter...
    - L'urgence sera plus importante que la sécurité
    - La sécurité sera plus importante que l'entretien
    - L'entretien sera plus importante que la cuisine

Encodeur de priorité

- Ce genre de chose s'implante bien...

A_3	A_2	A_1	A_0	MSB	LSB
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

A_3A_2 \ A_1A_0	00	01	11	10
00		1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

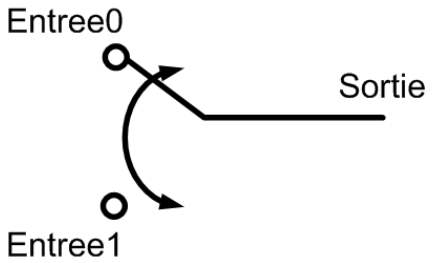
A_3A_2 \ A_1A_0	00	01	11	10
00		1	1	
01	1	1	1	
11	1	1	1	
10		1	1	

$$H_{MSB} = A_1 + A_0$$

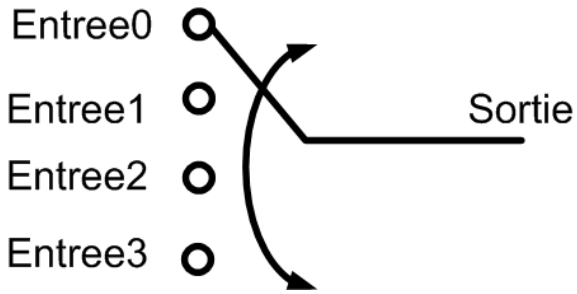
$$B_{LSB} = A_2 \overline{A_1} + A_0$$

Multiplexeur

- Un multiplexeur est un sélectionneur:
  - Il sélectionne quelle entrée va passer à la sortie
- On pourrait simplifier le diagramme et montrer quelque chose du style:



Multiplexeur 2 à 1

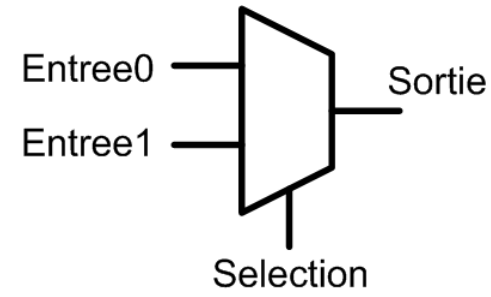


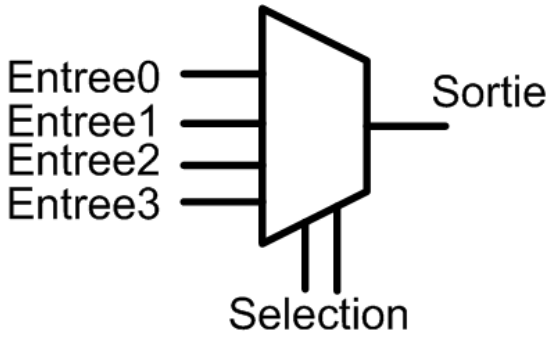
Multiplexeur 4 à 1

Évidemment, ce n'est pas aussi simple

Multiplexeur

- On utilise un signal de sélection pour indiquer quelle entrée passe à la sortie





Entree0	Entree1	Sel	Sortie
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

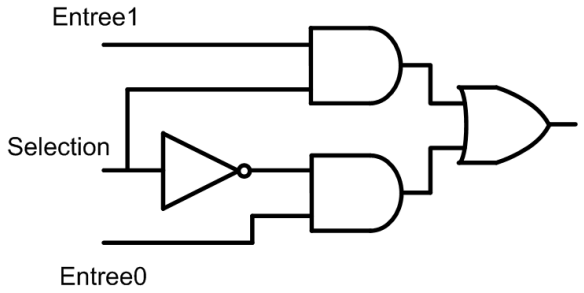
Multiplexeur

- Voici la table de verité d'un multiplexeur 2 à 1

SEL \ E_1E_0	00	01	11	10
0	0	1	1	0
1	0	0	1	1

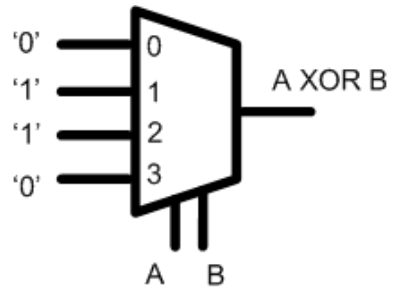
$$MUX = \overline{SEL} \bullet E_0 + SEL \bullet E_1$$

- Ça nous permet de trouver son circuit...



Fonctions avec multiplexeurs

- Tout comme les décodeurs, il est possible d'implanter des fonctions avec des MUX
- On peut implanter une fonction de N variables si notre MUX a N lignes de sélection
  - C'est à dire que notre MUX a 2^N entrées
- Ex: Pour 4 entrées, il y a 2 lignes de sélection
  - On peut implanter une fonction à 2 entrées avec un multiplexeur 4-à-1



Fonctions avec multiplexeurs

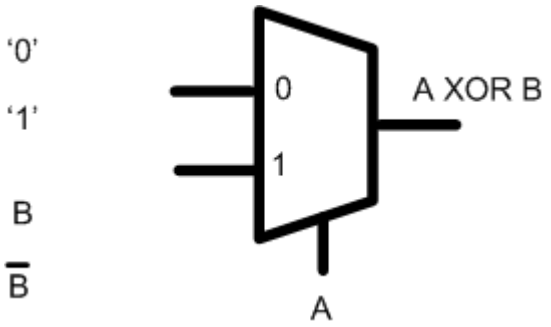
- Il est même possible d'implanter des fonctions à N entrées avec N-1 sélections
    - Ça fait que notre design sera plus efficace
- La dernière façon était facile:
    - On mettait soit '0' ou soit '1' selon la table de verité
- Cette fois, on va mettre N-1 entrées aux sélections
    - La dernière entrée sera parmi les entrées du MUX
    - Il faut jouer un peu avec pour que ça marche...

Exemple

- Concevez une porte XOR à 2 entrées avec un MUX 2-à-1.

Exemple

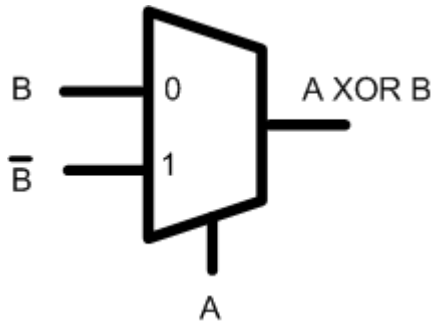
- On met une des entrées à la sélection et on voit ce qu'il manque:



  - Si A=0 et B=0, la sortie doit être 0
  - Ce 0 peut soit être un vrai 0, ou être égal àB
  - On ne le sait pas... continuons...
  - Si A=0 et B=1, la sortie doit être 1
  - On sait maintenant que l'entrée 0 doit être B

Exemple

- On continue le raisonnement:
  - Si A=1 et B=0, la sortie doit être 1
  - Ce 1 peut soit etre 1 ou l'inverse de B
  - On ne le sait pas... on continue...
  - Si A=1 et B=1, la sortie doit être 0
  - Ca ne peut etre que l'inverse de B
- Donc:

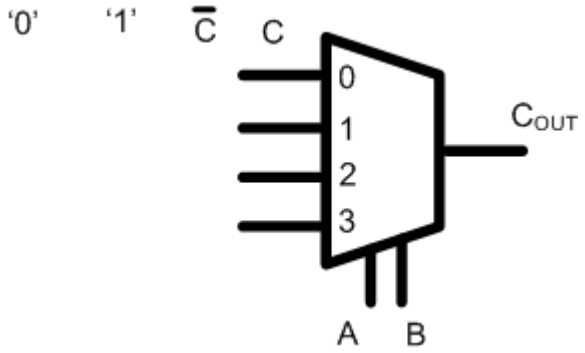


Exemple (seul)

- Implantez la fonction à 3 entrées qui calcule la retenue dans une addition
    - Utilisez un MUX 4-à-1

Exemple (seul)

- Pour 3 entrées, on a besoin d'un mux à 2 sélections (4 à 1 comme dans la question)
  - On va utiliser A et B dans la sélection et C sera aux entrées



A	B	C	C_{OUT}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Exemple (seul)

- Quand A=0 et B=0, on a toujours 0.
  - Entree0 sera 0
- Quand A=0 et B=1, la sortie est égale à C.
  - Entree1 sera donc connecté à C
- Quand A=1 et B=0, la sortie est égale à C
  - Entree2 sera aussi connecté à C
- Quand A=1 et B=1, la sortie est toujours 1.
  - Entree3 sera 1

Exemple (seul)

- Le circuit final ressemblera à ceci: