

Notions d'algèbre de Boole

Dans les systèmes digitaux (systèmes informatiques et autres automatismes numériques) toutes les données sont traitées et enregistrées à partir d'éléments d'informations binaires.

Ces informations binaires à la manière des contacts électriques n'ont que deux états possibles : un contact électrique est ouvert ou fermé, de même le bit est une information élémentaire qui ne peut prendre que deux valeurs 0 et 1.

Les opérations logiques sont en informatique aussi courantes si pas plus que les opérations arithmétiques. La logique combinatoire tout comme l'arithmétique repose sur quelques opérations élémentaires.

- En arithmétique, ces opérations sont l'addition, la soustraction, la multiplication et la division (+ , - , * , /). Il est possible à partir de là d'imaginer toutes les autres opérations telles que les exposants, les racines, les logarithmes etc.
- En logique, les opérations fondamentales sont le **ET**, le **OU** et le **NON**.
Nous utiliserons des signes particuliers pour représenter ces trois opérations fondamentales lors d'écriture d'équations logiques. C'est **George Boole**, un mathématicien britannique, qui le premier eu l'idée de reprendre des notations algébriques pour créer les bases de ce qui sera la logique informatique. Nous ferons donc de la **logique booléenne** et aussi de l'**algèbre booléenne** en écrivant des **équations logiques** pour exprimer les relations entre les variables logiques appelées aussi **variables booléennes**.

Cette logique a trouvé après George Boole ses premières applications dans les circuits électriques. C'est **Claude Shannon**, un autre père fondateur des théories à la base de l'informatique, qui entreprit de mettre en équation les circuits électriques où des relais électriques considérés comme des variables logiques en agissent sur des contacts ouverts (0) ou fermé (1).

La manière la plus simple de comprendre les fonctions logiques est de se les représenter par des schémas électriques qui comportent un ou plusieurs contacts et une lampe. Cette lampe s'allume "à condition" que les contacts électriques y laissent passer le courant. C'est dans l'expression de cette condition que va intervenir la logique.

Le schéma ci-contre traduit la condition la plus simple : La lampe s'allume si le bouton poussoir A est actionné. Autrement dit ($S = 1$) si ($A = 1$)

Le fonctionnement de ce circuit s'exprime par l'**équation** logique **$S = A$**

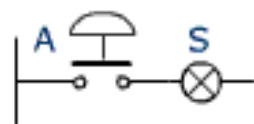


Fig. 1

Rendez-vous à la page <https://dcaclab.com/sl/lab> puis réalisez le montage en simulation.



Remarque : Vous savez qu'un circuit électrique ne fonctionne que s'il est fermé. Le retour au générateur n'est pas figuré sur le modèle Fig.1 par simplification mais est indispensable dans ce simulateur.

Collez ici une copie partielle d'écran de votre montage :

Dans la **table de vérité** d'une équation logique, on représente tous les cas possible pour la (ou les) entrées (s) et les résultats sur la (ou les) sorties. En mathématique, on parlerait d'antécédents et d'images

Ici il n'y a que deux cas possibles car une seule entrée A :

- bouton relevé (0) => lampe éteinte (0)
- bouton poussé (1) => lampe allumée (1)

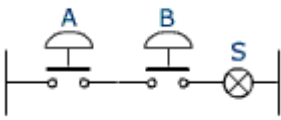
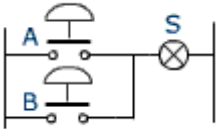
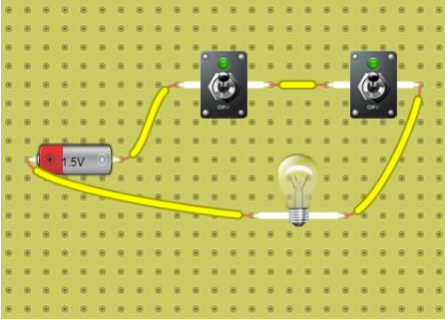
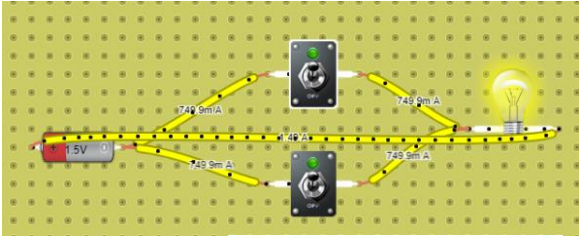
A	S
0	0
1	1

Une table de vérité a pour le rôle de montrer la correspondance entre la sortie et toutes les combinaisons de valeurs que peuvent prendre la ou les entrées.

Le contenu de la table de vérité s'obtient en imaginant toutes les configurations possibles pour le contact A (0 = contact relâché, 1 = bouton pressé). A chacun des états du contact A on vérifie d'après le schéma électrique ou d'après l'équation logique si la lampe est éteinte (0) ou allumée (1).

Fonctions ET et OU

Plaçons maintenant deux contacts dans le circuit (soit deux entrées A et B), ce qui va conduire à 4 situations différentes : aucun poussé, seulement A poussé, seulement B poussé, les deux poussés. On peut de plus imaginer les deux cas suivants

schéma																																						
analyse	La lampe s'allume si A est poussé ET B est poussé	La lampe s'allume si A est poussé OU B est poussé																																				
Table de vérité	 <table border="1"> <thead> <tr> <th colspan="2">Entrée</th> <th>Sortie</th> </tr> <tr> <th>a</th> <th>b</th> <th>c</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> <p>Fig. 3. - Table de vérité du circuit ET.</p>	Entrée		Sortie	a	b	c	0	0	0	0	1	0	1	0	0	1	1	1	<p>Collez ici la tabl</p>  <table border="1"> <thead> <tr> <th colspan="2">Entrées</th> <th>Sorties</th> </tr> <tr> <th>a</th> <th>b</th> <th>L</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> <p>e de vérité de la fon ction OU</p>	Entrées		Sorties	a	b	L	0	0	0	0	1	1	1	0	1	1	1	1
Entrée		Sortie																																				
a	b	c																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
Entrées		Sorties																																				
a	b	L																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	1																																				
équation	$S = A.B$	$S = A + B$																																				

Sur le même simulateur que ci-dessus, réalisez les montages afin d'établir les deux tables de vérité demandées (ou les vérifier si vous les avez établies sans recourir au test)

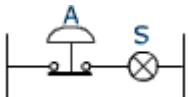
Remarque : vous devrez utiliser les clics droit et gauche de la souris pour « appuyer » simultanément sur les des boutons A et B

L'opérateur **ET** est représenté dans l'équation logique par un point. Ce signe convient parfaitement puisque la fonction ET donne le même résultat qu'une multiplication.

L'opérateur **OU** est représenté dans l'équation logique par un signe plus. La fonction OU donne le même résultat qu'une addition (sauf pour 1 + 1) c'est pourquoi on écrit parfois + surmonté d'un point

Fonction NON

Les contacts que nous avons utilisés jusqu'ici, sont des contacts "normalement ouverts". Quand le bouton poussoir est relâché (quand A = 0) le courant ne passe pas. Nous utilisons maintenant un contact "normalement fermé" pour illustrer la fonction NON. Au repos, le courant passe mais il se coupe quand le contact est activé (quand A = 1)

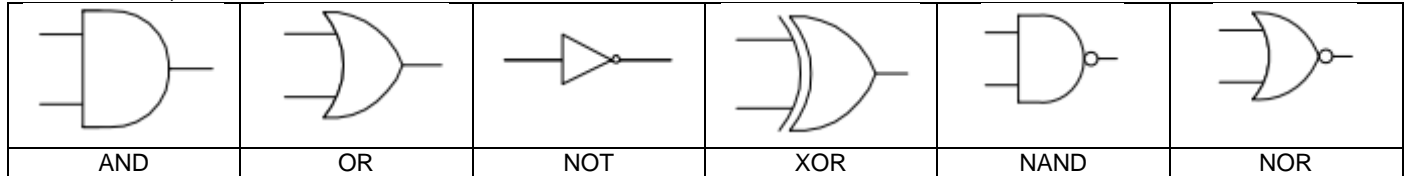
	<p>Les deux cas :</p> <ul style="list-style-type: none">- bouton relevé (0) => lampe allumé (1)- bouton poussé (1) => lampe éteinte (0)	<table><tr><th>A</th><th>S</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	S	0	1	1	0	<p>$S = \bar{A}$</p> <p>Le signe 'non' est la barre au-dessus</p>
A	S								
0	1								
1	0								

Les portes logiques

Nous avons jusqu'ici utilisé des boutons poussoirs et une lampe pour illustrer le fonctionnement des opérateurs logiques. En électronique digitale, les opérations logiques sont effectuées par des **portes logiques**. Ce sont des circuits de très petite taille implantés en très grand nombre sur des puces de silicium et qui combinent les signaux logiques présentés à leurs **entrées** sous forme de tensions. On aura par exemple 5V pour représenter l'état logique 1 et 0V pour représenter l'état 0.

Dans un microprocesseur moderne on compte plusieurs dizaines de milliards de portes logiques. Selon leur type, il faut de 2 à 10 transistors (composant électronique de base des portes logiques) pour réaliser chaque porte logique.

Ci-dessous les schémas des portes logiques courantes avec leurs entrées ('pattes' situées à gauche) et leurs sorties ('pattes' situées à droite)



AND, OR, NOT n'ont plus de secret pour vous

XOR est le OU EXCLUSIF. Il répond comme OR sauf quand les deux entrées sont à 1 : sa sortie est alors à 0.

illustration : Un établissement de soin accueille des personnes âgées **OU** malades : c'est le OR

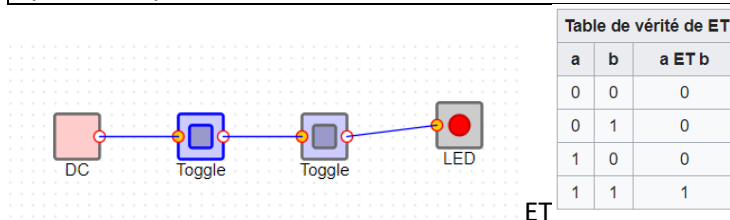
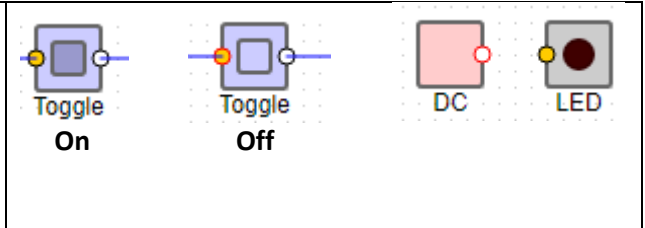
Un restaurateur vous propose fromage **OU** dessert : c'est le XOR

NAND (pour Not AND) et NOR (pour Not OR) sont les AND et OR suivies de NON.

Elles répondent exactement le contraire de AND et OR.

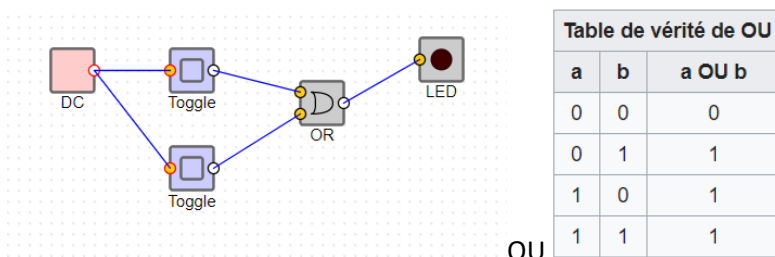
Rendez-vous à la page <file:///E:/exercices%20NSI/simcirjs-master/simcirjs-master/sample.html> Ce simulateur vous propose toutes ces portes logiques. Câblez et vérifiez les tables de vérité des fonctions AND et OR Etablir les tables de vérité des fonctions XOR, NAND, NOR

Remarque : Evitez les boutons PuschOn au profit des Toggle (bouton à bascule) qui ont l'avantage de rester dans l'état ou vous les basculez
Ici le bouclage des cablages n'est pas représenté c'est pourquoi une alimentation (DC) ou une LED ne comporte qu'un seul pôle



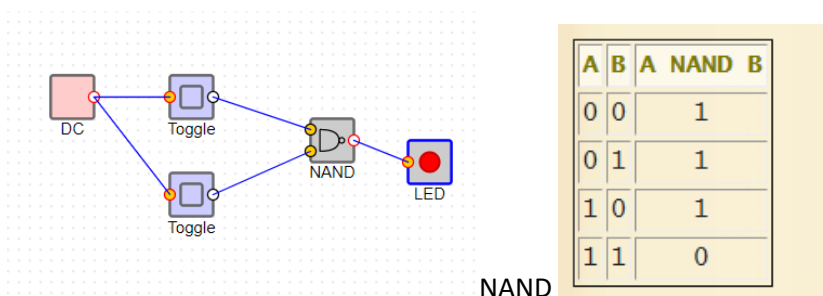
ET

a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1



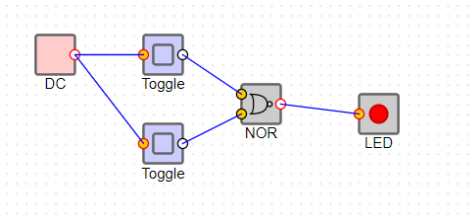
OU

a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1



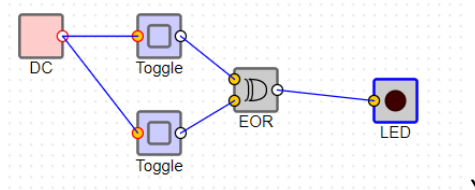
NAND

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



NOR

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



XOR

Table de vérité de XOR (OU exclusif)		
a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Mais quel rapport avec les ordinateurs ?

Nous allons répondre à cette question en nous limitant à l'opération la plus simple que nous demandons à un ordinateur : réaliser la somme de deux entiers.

C'est la plus simple mais d'elle découlent :

- les différences (même processus avec un entier signé négatif)
- certaines multiplications (sommes répétitives)

Poser verticalement et calculer en binaire la somme des deux entiers $A = (14)_{10}$ et $B = (9)_{10}$

A et B sont appelés **opérandes**

1110 A
1001 B
10111 A+B

Qu'avez-vous fait ?

Commençons par la colonne de droite (bit de 2^0 de poids faible) : vous avez réalisé une opération logique :

- si les deux étaient à 0 vous avez écrit en dessous pour la somme **0**
- si un des deux étaient à 0 et l'autre à 1 vous avez écrit en dessous pour la somme **1**
- si les deux étaient à 1 vous avez écrit en dessous pour la somme **0** et vous avez pensé à **retenir 1**

Ecrire ci-dessous les tables de vérité des deux opérations logiques qui donnent (pour le bit de poids faible)

- la **somme** à partir des deux opérandes
- la **retenue** (0 si elle n'existe pas, 1 si elle existe)

X = a b		
a	b	X
0	0	0
0	1	1
1	0	1
1	1	1

Table de vérité de ET		
a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

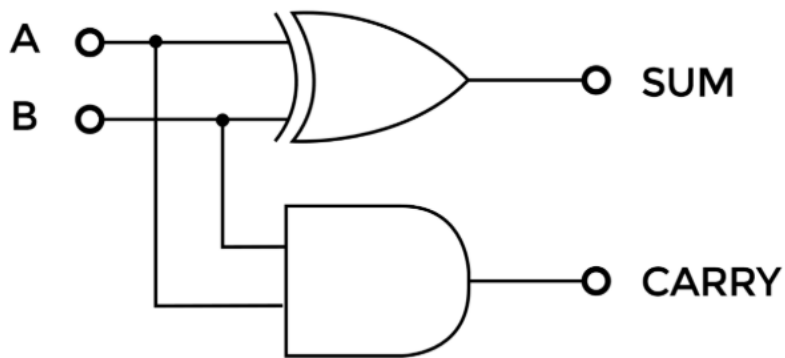
SOMME → OU

RETENUE → ET

Recherchez plus haut quelles portes logiques étudiées effectuent exactement ces mêmes tâches.

Recherchez sur internet une image correspondant au mot clé 'half adder' ou 'demi additionneur' le schéma doit confirmer votre réponse précédente. Quel mot anglais se cache dans ces schémas derrière la lettre C (ou C_{out}) ?

Pourquoi parle-t-on d'une fonction logique à 2 entrées et 2 sorties ?



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

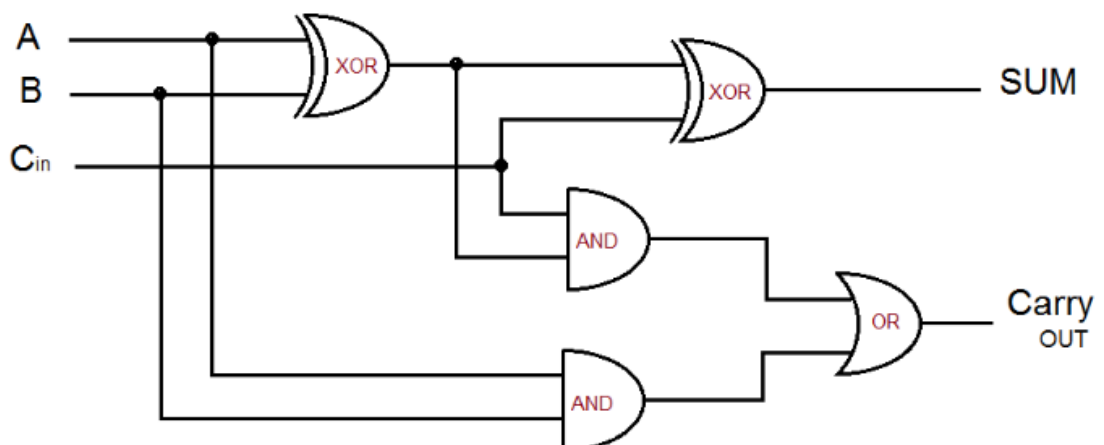
Le *half adder* ajoute deux bits ensemble. Le demi additionneur a deux signaux d'entrée représentant des chiffres binaires (a et b) et deux signaux de sortie, dont l'un est le résultat de l'addition (s), et l'autre le **carry** en classe supérieure (C). Il est important de noter qu'un demi additionneur ne peut pas être utilisé pour ajouter des nombres binaires à plusieurs chiffres parce qu'il n'y a pas de port de niveau inférieur. Le demi additionneur est un circuit combiné de circuits XOR et AND. Son but, comme son nom l'indique, est d'ajouter des chiffres. Le processus d'addition de nombres dans le système binaire est réduit à l'addition de chiffres, où l'on obtient ainsi une somme et un carry. Puisque le demi additionneur lui-même ne peut pas calculer le résultat entier, il est combiné avec un autre demi additionneur et un circuit OU pour faire un additionneur complet.

<http://www.differencebetween.net/technology/difference-between-half-adder-and-full-adder/>

A partir de la deuxième colonne (bits de 2^1 puis suivants)

- la prise en compte des deux bits en provenance de A et B ne suffit plus, il faut aussi considérer la **retenue**. La fonction logique '*half adder*' ne suffit plus, on a besoin de la fonction '*full adder*'. Cherchez sur internet un schéma en portes logiques de '*full adder*' (ou '*plein additionneur*'). Il était un peu compliqué à deviner !

Full Adder Logic Gate Diagram

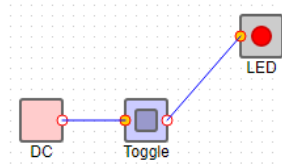


A l'aide du simulateur <file:///E:/exercices%20NSI/simcirjs-master/simcirjs-master/sample.html> nous allons câbler un full adder, vérifier qu'il fonctionne correctement et établir sa table de vérité.

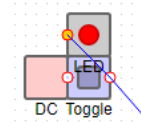
Préliminaire : Construction d'entrées sélectionnables (0/1) par bouton (Toggle) puis regroupement sur un petit espace

Alimentation, bouton de choix et LED raccordés entre eux

=>



une fois regroupé =>



Ce petit 'pavé' constitue une **entrée** avec sa diode témoin, celle-ci n'est pas indispensable mais aide à la visualisation de l'état de l'entrée (0 ou 1). Ce pavé est à raccorder aux portes logiques à partir de la borne de sortie du Toggle. Vous répétez cette petite construction pour constituer autant d'entrées que nécessaires.

Pour visualiser les **sorties**, il suffira bien évidemment de raccorder une DEL

A partir du schéma que vous avez trouvé, réalisez sur le simulateur le montage de l'additionneur plein avec ses entrées, la combinaison des portes logiques, les sorties.

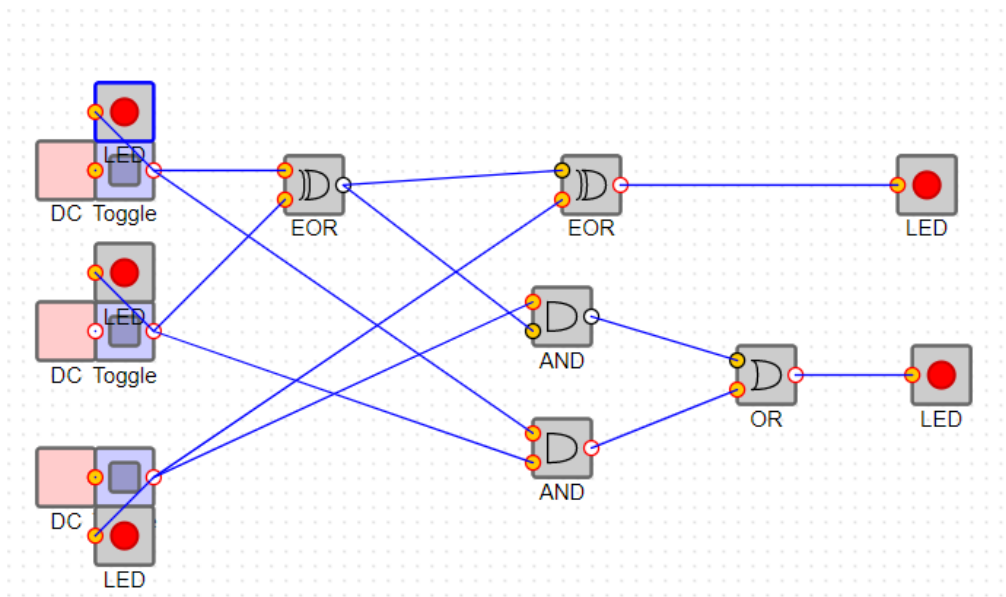
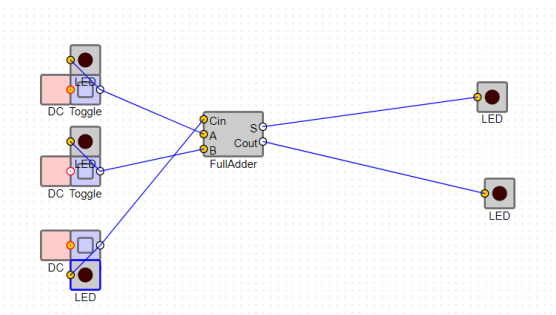


Table de vérité complète que vous avez établie à l'aide de votre montage (5 colonnes, 8 lignes)

Full Adder Truth table

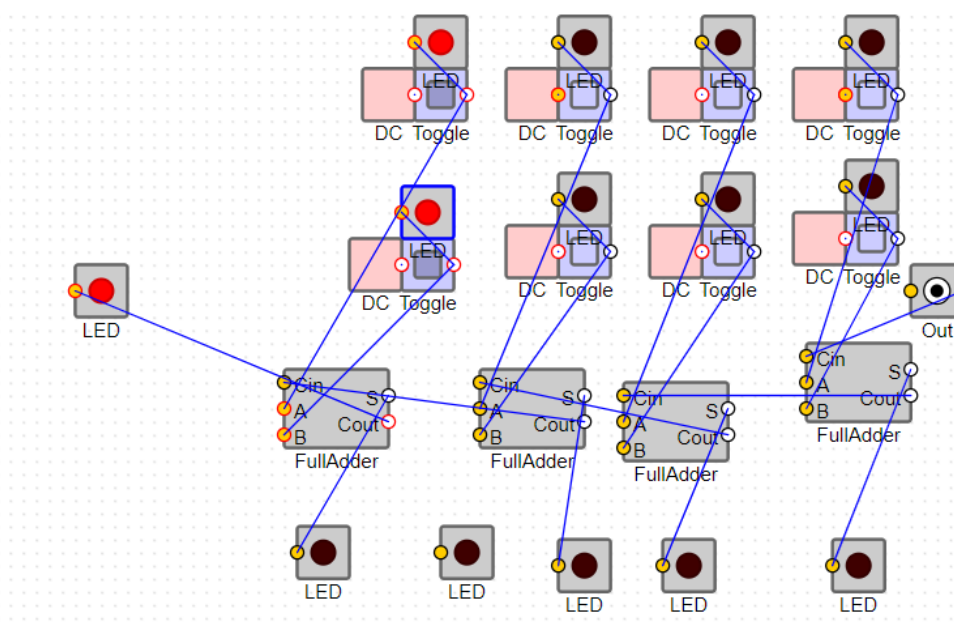
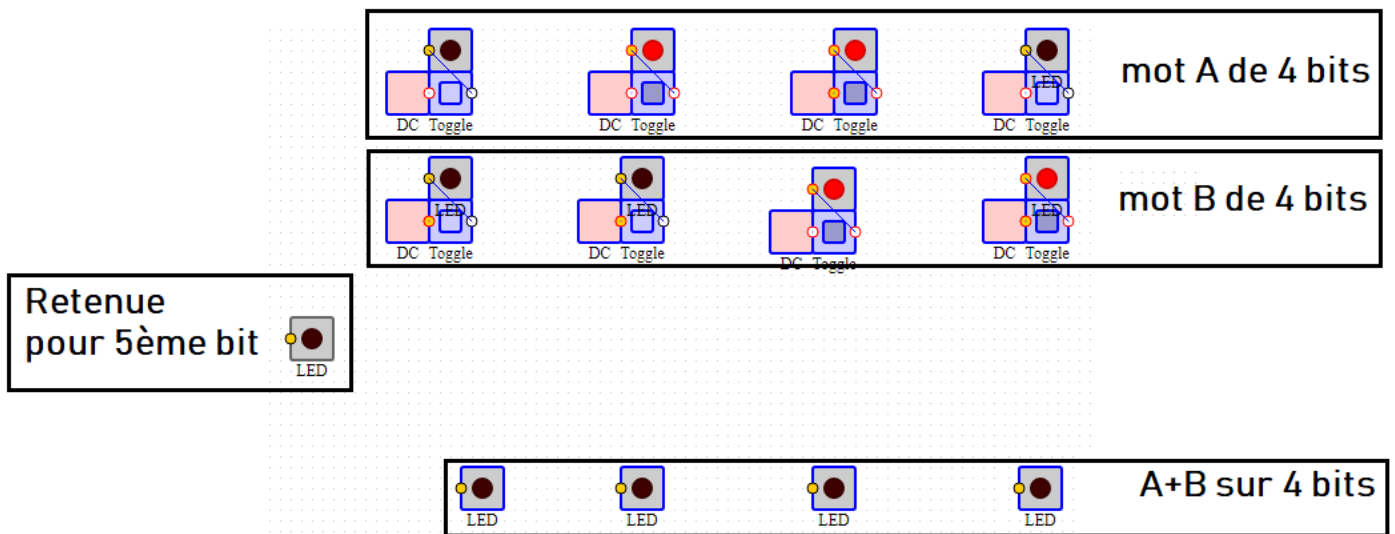
INPUTS			OUTPUTS	
A	B	C _{in}	SUM	CARRY _{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Le simulateur propose la fonction Full adder toute faite. Sur votre schéma précédent conservez les entrées et les sorties et retirez toutes les portes logiques. Remplacez cet ensemble de porte logique par un 'full adder', recâblez, vérifiez votre table de vérité.



Réalisation d'un additionneur 4 bit

Sur l'image ci-dessous, on a représenté une addition posée verticalement. Recopiez ce schéma sur le simulateur et câblez dans l'espace libre les 'full adder' nécessaires pour les additions se fassent correctement quels que soient les entrées choisies



Pour aller plus loin : Quelle fonction est réalisée par ce montage à 6 entrées et 1 sortie ?

On considérera qu'il admet 2 entrées A et B constitués chacun d'un mot de 3 bits.

Il est constitué de 3 portes XNOR (Not XOR) et d'une porte AND à 3 entrées (celle-ci n'est pas disponible dans le simulateur, vous devrez trouver une solution)

