

## A. Les booléens :

### Ce qu'il faut savoir :

- **George Boole**, un mathématicien et philosophe britannique du XIX<sup>e</sup> siècle, créa de 1844 à 1854 une algèbre binaire, dite booléenne, n'acceptant que deux valeurs numériques : 0 et 1.

Cette algèbre aura de nombreuses applications en téléphonie et en informatique.



George Boole

- Un booléen est un type de variable (« **bool** ») qui n'a que deux états qui sont en général **True** (représenté par un bit 1) ou **False** (représenté par un bit 0).
- Sur Python, il ne faut pas confondre le symbole « **=** » qui sert à affecter une valeur à une variable avec le symbole « **==** » qui sert à tester une égalité.

Exemple : Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a=b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on affecte la valeur de *b* à *a* .  
(à la fin *a* vaut 3)

Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a==b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on teste si la valeur de *a* est égale à celle de *b* .  
(ici *a == b* est un booléen qui a la valeur False)

- Sur Python, « **==** » est un opérateur de comparaison.  
Les autres opérateurs de comparaison sont : « **!=** » (différent de), « **<** » (strictement inférieur), « **>** » (strictement supérieur), « **<=** » (inférieur ou égal) et « **>=** » (supérieur ou égal).

**Activité 1 :** Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

1. >>>a=3  
>>>b=5  
>>>a>b

False

2. >>>a=3  
>>>b=5  
>>>a!=b

True

3. >>>type(5)==str

False

4. >>>5=="5"  
False

5. >>>5==5.0  
True

6. >>>type(5)==type(5.0)  
False

7. >>>var1="vive la nsi"  
>>>var2="nsi"  
>>>var2 in var1  
True

8. >>>var1="vive la nsi"  
>>>var2="nsi"  
>>>var1 in var2  
False

```

9. >>>a="j'aime"
   >>>b=" la nsi"
   >>>c=a+b
   >>>c=="j'aime la nsi"
      True

10. >>>a=-5.0
    >>>b=4.2
    >>>a*b>4*a-1
      False

```

## Ce qu'il faut savoir : Les opérateurs booléens.

- L'opérateur « **not** » ( qui signifie « non ») est le booléen **opposé**.

Voici **la table de vérité** qui définit cet opérateur pour un booléen a :

a	False	True
not a	True	False

a	0	1
not a	1	0

Remarque : quand le booléen est défini avec les chiffres 0 et 1, alors **not a = 1 - a**.

- L'opérateur « **and** » ( qui signifie « et ») teste si deux booléens sont vrais **simultanément**.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a and b	False	True
False	False	False
True	False	True

a and b	0	1
0	0	0
1	0	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a and b = a x b**.

- L'opérateur « **or** » ( qui signifie « ou ») teste si **au moins un des deux** booléens est vrai.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a or b	False	True
False	False	True
True	True	True

a or b	0	1
0	0	1
1	1	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a or b = a + b - a x b**.

**Activité 2 :** Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

- |  |  |   |  |
|--|--|---|--|
| <p>1. &gt;&gt;&gt;(8&gt;4 or 1&gt;2)</p> <p style="text-align: center;">True</p>                           | <p>2. &gt;&gt;&gt;a=True</p> <p>&gt;&gt;&gt;b=False</p> <p>&gt;&gt;&gt;not(a or b) or (a and not(b))</p> <p style="text-align: center;">True</p> | <p>3. &gt;&gt;&gt;a=3</p> <p>&gt;&gt;&gt;b=-7</p> <p>&gt;&gt;&gt;a**3&gt;50 and b**2&lt;50</p> <p style="text-align: center;">False</p> | <p>4. &gt;&gt;&gt;a=3</p> <p>&gt;&gt;&gt;b=-7</p> <p>&gt;&gt;&gt;(a**3&gt;50 and b**2&lt;50)</p> <p>or (a**2&lt;10 and b**2&gt;10)</p> <p style="text-align: center;">True</p> |
| <p>5. &gt;&gt;&gt;mot="science"</p> <p>&gt;&gt;&gt;mot[0]=="s"</p> <p style="text-align: center;">True</p> | <p>6. &gt;&gt;&gt;mot="science"</p> <p>&gt;&gt;&gt;mot[0]=="s" and mot[5]=="n"</p> <p style="text-align: center;">False</p>                      | <p>7. &gt;&gt;&gt;mot="science"</p> <p>&gt;&gt;&gt;not(mot[2:5]=="ien")</p> <p style="text-align: center;">False</p>                    | <p>8. &gt;&gt;&gt;var1=" vive la nsi"</p> <p>&gt;&gt;&gt;var2="vive le prof"</p> <p>&gt;&gt;&gt;"vive" in (var1 and var2)</p> <p style="text-align: center;">True</p>          |

## **B. Détermination des tables de certaines fonctions booléennes :**

1. **a.** Recopie et complète le tableau ci-dessous.

a	b	a and b	b and a	a or b	b or a
False	False	False	False	False	False
False	True	False	False	True	True
True	False	False	False	True	True
True	True	True	True	True	True

**b.** Que peux-tu en déduire pour les booléens (a and b) , puis (b and a) ? Et pour (a or b), puis (b or a) ?  
**(a and b)=(b and a)                      puis                      (a or b)=(b or a)**

2. Sur le même modèle, construire un tableau pour chacune des questions et en tirer des conclusions :

**a.** (a and True).                      **b.** (a or False) .

a	a and True	a or False
False	False	False
True	True	True

**c.** (a and (b or c)) puis ((a and b) or (a and c)).

a	b	c	a and (b or c)	(a and b) or (a and c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	False	False
True	False	False	False	False
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

**d.** (a or (b and c)) puis ((a or b) and (a or c)).

a	b	c	a or (b and c)	(a or b) and (a or c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	True	True
True	False	False	True	True
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

e. (a and not(a)).

f. (a or not(a)).

g. not(not(a)).

a	a and not(a)	a or not(a)	not(not(a))
False	False	True	False
True	False	True	True

h. ((a and b) and c) puis (a and (b and c)).

a	b	c	(a and b) and c	a and (b and c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	False	False
True	False	False	False	False
True	False	True	False	False
True	True	False	False	False
True	True	True	True	True

i. ((a or b) or c) puis (a or (b or c)).

a	b	c	(a or b) or c	a or (b or c)
False	False	False	False	False
False	False	True	True	True
False	True	False	True	True
False	True	True	True	True
True	False	False	True	True
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

j. (a and (a or b)) puis a.

k. (a or a and b) puis a.

a	b	a and (a or b)	a or a and b
False	False	False	False
False	True	False	False
True	False	True	True
True	True	True	True

l. not(a and b) puis (not(a) or not(b)).

a	b	not(a and b)	not(a) or not(b)
False	False	True	True
False	True	True	True
True	False	True	True
True	True	False	False

m.  $\text{not}(a \text{ or } b)$  puis  $\text{not}(a)$  and  $\text{not}(b)$

a	b	$\text{not}(a \text{ or } b)$	$\text{not}(a) \text{ and } \text{not}(b)$
False	False	True	True
False	True	False	False
True	False	False	False
True	True	False	False

3. Résume les informations découvertes à travers cet exercice (ce sont les théorèmes sur les booléens).

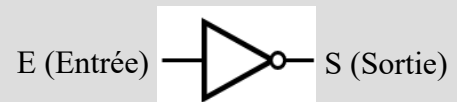
- $(a \text{ and } \text{True}) = (a \text{ or } \text{False}) = a$
- $(a \text{ and } \text{not}(a)) = \text{False}$        $(a \text{ or } \text{not}(a)) = \text{True}$        $\text{not}(\text{not}(a)) = a$
- $(a \text{ and } (b \text{ or } c)) = ((a \text{ and } b) \text{ or } (a \text{ and } c))$
- $(a \text{ or } (b \text{ and } c)) = ((a \text{ or } b) \text{ and } (a \text{ or } c))$
- $((a \text{ and } b) \text{ and } c) = (a \text{ and } (b \text{ and } c))$
- $((a \text{ or } b) \text{ or } c) = (a \text{ or } (b \text{ or } c))$
- $(a \text{ and } (a \text{ or } b)) = (a \text{ or } a \text{ and } b) = a$
- $\text{not}(a \text{ and } b) = (\text{not}(a) \text{ or } \text{not}(b))$
- $\text{not}(a \text{ or } b) \text{ puis } (\text{not}(a) \text{ and } \text{not}(b))$

### C. Portes logiques :

#### Ce qu'il faut savoir :

- Les circuits d'un ordinateur manipulent uniquement des chiffres binaires (0 et 1) qui, techniquement en interne, sont simplement représentés par des tensions électriques. Ainsi, le chiffre 0 est représenté par une tension basse proche de 0 Volt et le chiffre 1 par une tension haute qui dépend des circuits.
- Les composantes qui vont permettre de faire des opérations sur ces chiffres binaires sont appelées **portes logiques**. Ces portes logiques sont des circuits électroniques composés de **transistors** qui se comportent comme **des interrupteurs qui laissent passer ou non le courant électrique**.
- Les portes logiques élémentaires sont :

- la porte « not »



- la porte « and »



- la porte « or »



### Activité 1 :

Les entrées et sorties sont des tensions électriques représentant des booléens 0 (pour False) et 1 (pour True).

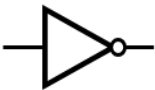
1. On veut créer une porte logique « not » sur le logiciel Logisim et simuler son fonctionnement.


Pour cela observe la vidéo suivante : <https://www.youtube.com/watch?v=gCOsSLtMogo>


Ouvre le logiciel Logisim et réalise ce travail.

2. Sur la même page du logiciel Logisim, crée une porte logique « and », puis une porte logique « or » et simule leur fonctionnement.

3. Complète alors les tables de vérité ci-dessous :

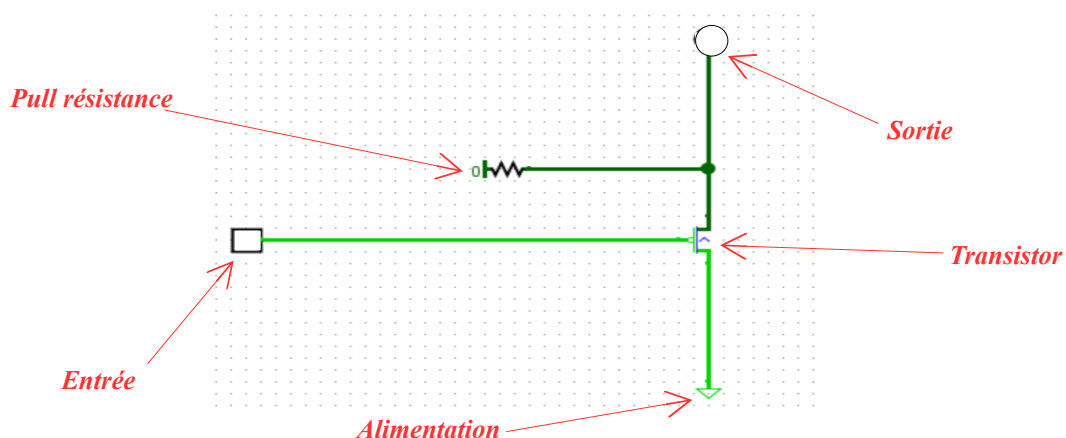
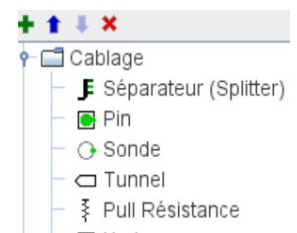
	
not	
E	S
0	1
1	0


		
and		
E1	E2	S
0	0	0
0	1	0
1	0	0
1	1	1

		
or		
E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	1

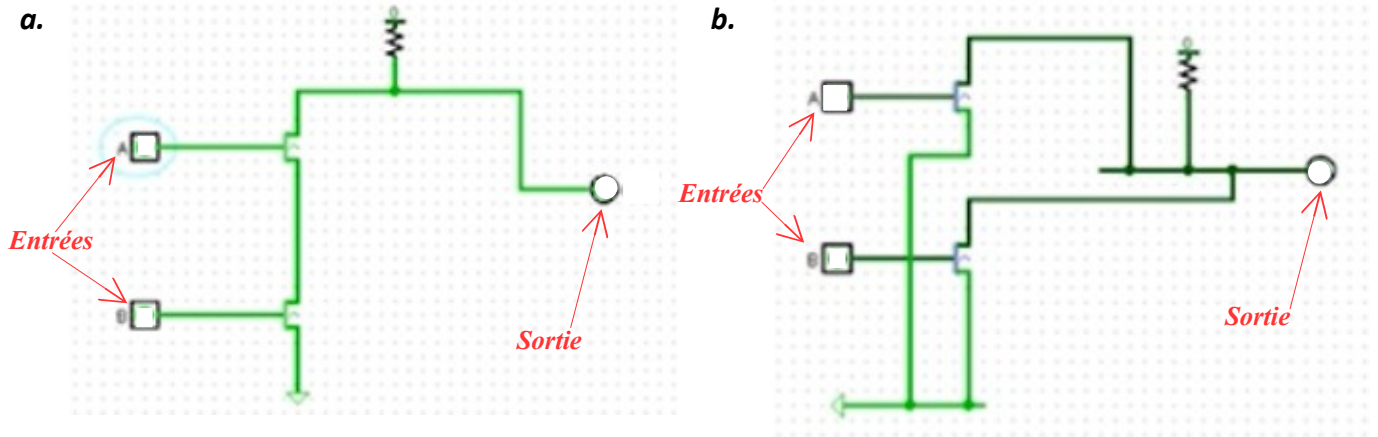
**Activité 2 :** On veut maintenant construire ces 3 portes logiques à l'aide de transistors avec le logiciel Logisim.

1. a. Ouvre un nouveau fichier Logisim et réalise le circuit ci-dessous en utilisant les composants accessibles dans le dossier « câblages » visible ci-contre et accessible en haut à gauche du fichier Logisim.



- b. A l'aide de cet outil  Simule le fonctionnement de ce circuit en mettant l'entrée à 0, puis 1.  
Quelle porte logique reconnaît-on ?
- c. En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre ton travail dans le dossier **Partage (R) : NSI : Gauthier** en lui donnant le nom : « **NOM.fichier1** ».

2. Réalise de même chacun des deux circuits ci-dessous et précise à quelles porte logique correspond chacun d'entre eux.



c. En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre tes travaux dans le dossier **Partage (R) : NSI : Gauthier** en leur donnant le nom : « **NOM.fichier2** » et « **NOM.fichier3** ».

**Activité 3 :** On veut additionner 2 nombres binaires de 1 bit. Pour cela on va construire 2 fonctions logiques S (pour Somme) et R (pour Retenue) qui prennent en entrées E1 et E2 les 2 nombres binaires.

1. Recopie et complète la table de vérité de chacune de ces fonctions logiques :

<b>SOMME</b>		
E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	0

<b>RETENUE</b>		
E1	E2	R
0	0	0
0	1	0
1	0	0
1	1	1

2. a. A quel opérateur correspond la fonction logique qui donne la retenue ? **And**

b. Pour la fonction logique qui donne la somme, dans quels cas S est à 1 (True) ?

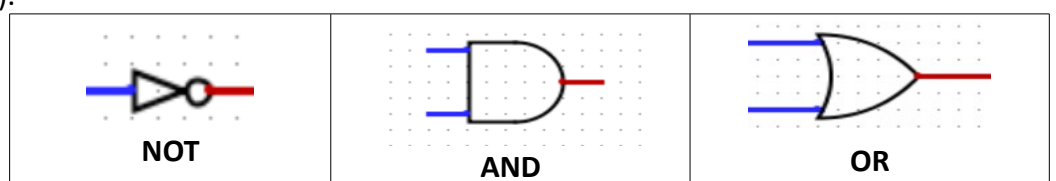
**Quand on a : (faux pour E1 et vrai pour E2) ou (vrai pour E1 et faux pour E2)**

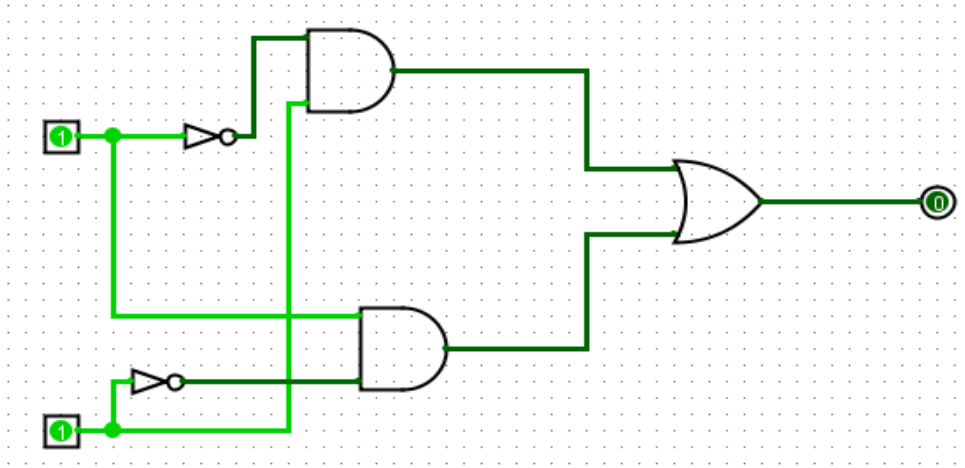
c. Partant de E1=True (1) et E2=True (1), écris une expression logique avec E1 et E2 traduisant ta réponse du b.

**(not(E1) and E2) ou (E1 and not(E2))**

d. A l'aide de ce qui vient d'être fait, complète chaque « ? » par une des portes logiques utilisables (données ci-dessous) ce circuit dans lequel on a 2 entrées (les nombres binaires E1 et E2) et la sortie S (la somme).

Portes logiques utilisables  
(une ou plusieurs fois) :



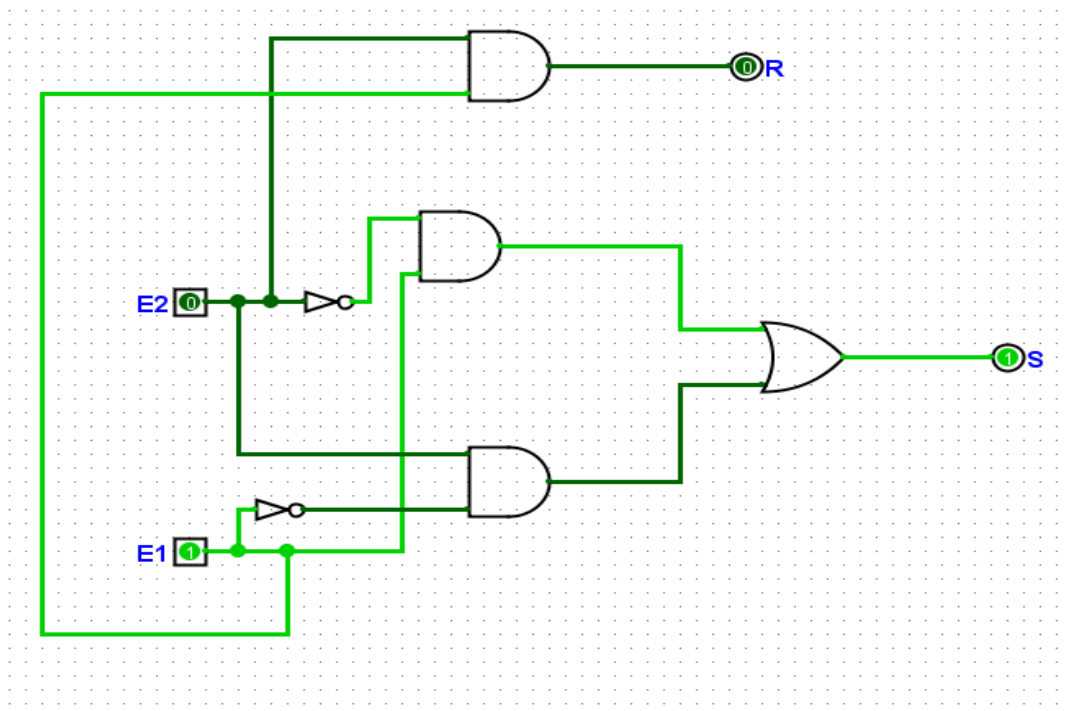


- e. Ouvre un nouveau fichier Logisim, puis à l'aide des questions précédentes, reproduis le circuit de la question c. et complète le afin qu'il permette d'obtenir les résultats des fonctions logiques S (Somme), mais aussi R (Retenue). Ce circuit se nomme **un demi-additionneur sur 1 bit.**

Par un clic sur le circuit nommé « main » en haut à gauche de la page, fait apparaître le nom « **main** » du circuit, que tu vas remplacer par « **demi\_add** ».

Veille à faire apparaître E1, E1 et S pour les entrées et la sortie en cliquant dessus et en inscrivant ces noms dans le label.

VHDL	Verilog
Représentation	Est
Emplacement du label	Nord
Police du Label	SansSerif Gras 16
Label Visible	Oui
Nom du circuit	main
Label partagé	





3. On veut maintenant additionner 2 nombres binaires de 1 bit en tenant compte de la retenue entrante. Pour cela on va construire 2 fonctions logiques  $S$  (pour Somme) et  $R_s$  (pour Retenue sortante) qui prennent en entrées  $E1$  et  $E2$  les 2 nombres binaires, et  $R_e$  (pour retenue entrante).

Recopie et complète la table de vérité de chacune de ces fonctions logiques :

<b>SOMME</b>			
$R_e$	$E1$	$E2$	$S$
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

<b>RETENUE sortante</b>			
$R_e$	$E1$	$E2$	$R_s$
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

4. a. On va commencer par additionner (avec un demi-additionneur)  $E1$  et  $E2$  et on obtient une première somme intermédiaire  $S_i$  et une première retenue intermédiaire  $R_i$ . Il faut évidemment encore additionner (avec un autre demi-additionneur)  $S_i$  et  $R_e$  afin d'obtenir une seconde somme intermédiaire  $s_i$  et une seconde retenue intermédiaire  $r_i$ .

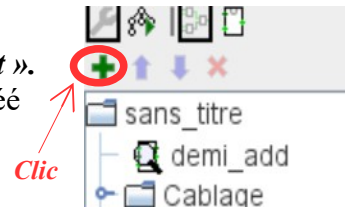
Recopie et complète le tableau ci-dessous :

$R_e$	$E1$	$E2$	$S_i$	$R_i$	$s_i$	$r_i$
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	0	0	1	0
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	0

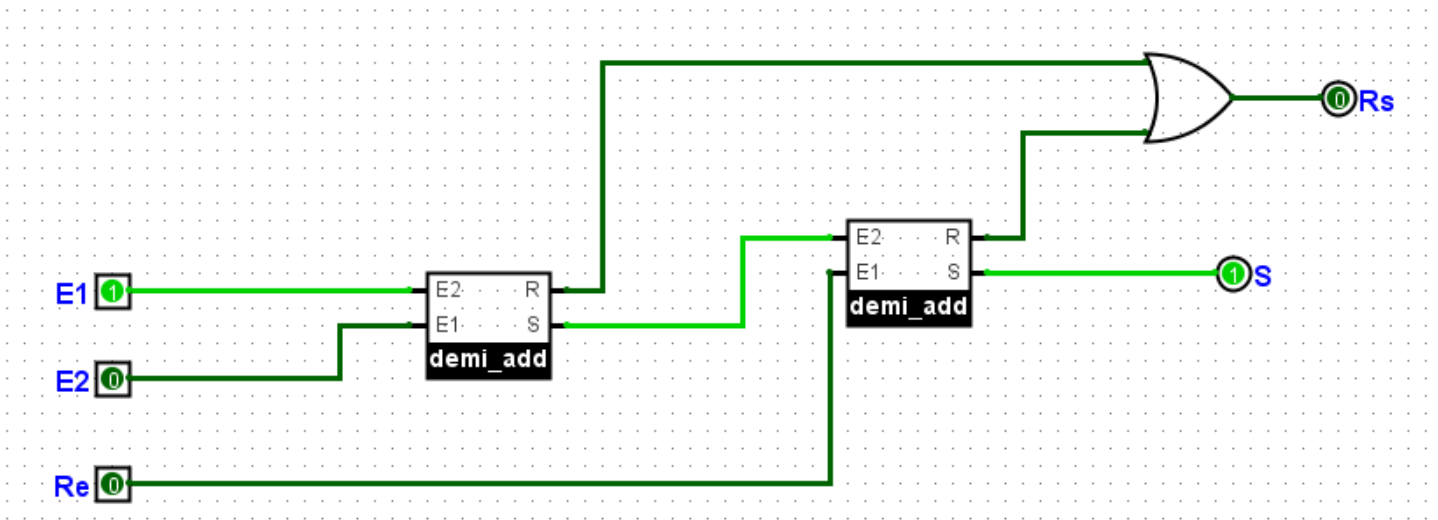
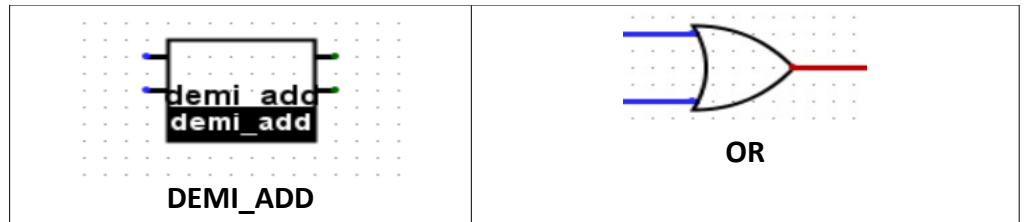
b. En observant attentivement les deux derniers tableaux, compare la seconde somme intermédiaire  $s_i$  et la SOMME cherchée. Puis trouve un lien logique entre  $R_i$ ,  $r_i$  et la RETENUE SORTANTE cherchée.

$s_i = \text{SOMME}$  et  $R_i \text{ or } r_i = \text{RETENUE SORTANTE}$

c. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « *add\_complet* ». Puis complète le circuit ci-dessous où l'on a utilisé le circuit « *demi\_add* » créé à la question 2.

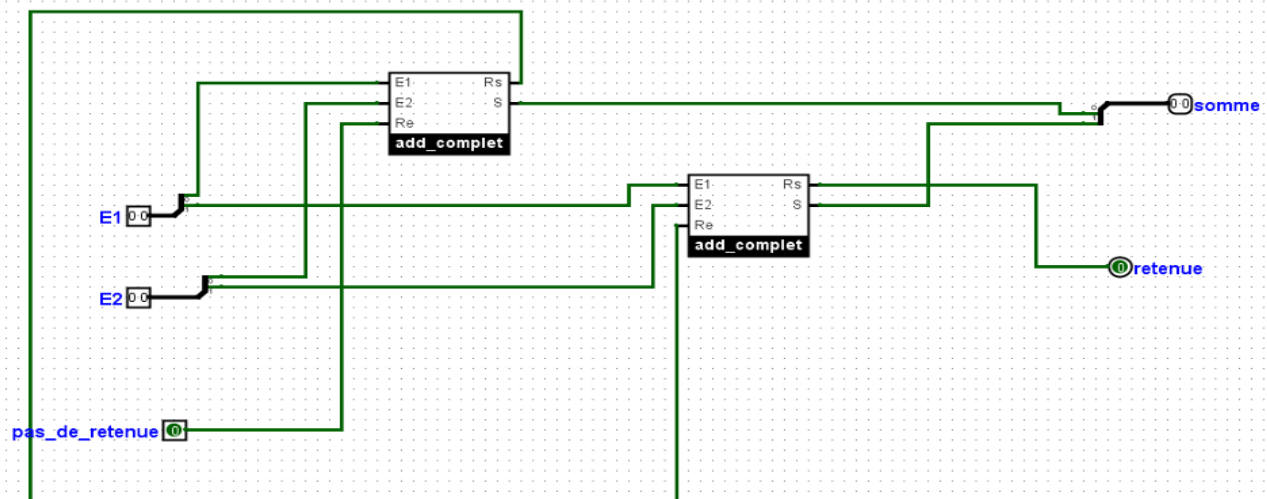


éléments utilisables  
(une ou plusieurs fois) :



d. A l'aide des questions précédentes, reproduis sur Logisim ce circuit dans le circuit « *add\_complet* ». Ce circuit se nomme un additionneur complet sur 1 bit.

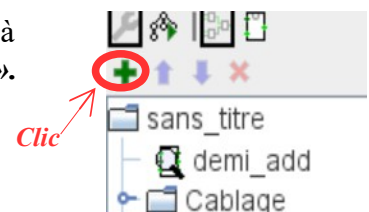
5. On s'intéresse maintenant au circuit cidessous, où E1, E2 et S représentent des nombres binaires de 2 bits:



a. Explique en quelques phrases le fonctionnement de ce circuit.

Avec un premier « *add\_complet* » on additionne les premiers bits de E1 et E2 en prenant 0 comme retenue entrante (car pas de retenue au départ du calcul). La somme obtenue est le premier bit de la somme en sortie et la retenue sortante devient la retenue entrante du second « *add\_complet* ». Dans le second « *add\_complet* » on additionne les seconds bits de E1 et E2 en prenant comme retenue entrante, la retenue sortante précédente. La retenue obtenue est la retenue en sortie et la somme obtenue est le second bit de la somme en sortie.

b. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « *add\_2bits* ». Puis reproduis le circuit précédent.



c. Teste ce circuit et reporte les résultats de tests dans le tableau ci-dessous que tu compléteras.

E1	E2	somme	retenue
(0,0)	(0,0)	(0,0)	0
(0,0)	(1,0)	(1,0)	0
(0,0)	(0,1)	(0,1)	0
(0,0)	(1,1)	(1,1)	0
(0,1)	(0,0)	(0,1)	0
(0,1)	(1,0)	(1,1)	0
(0,1)	(0,1)	(1,0)	0
(0,1)	(1,1)	(0,0)	1
(1,0)	(0,0)	(1,0)	0
(1,0)	(1,0)	(0,0)	1
(1,0)	(0,1)	(1,1)	0
(1,0)	(1,1)	(0,1)	1
(1,1)	(0,0)	(1,1)	0
(1,1)	(1,0)	(0,1)	1
(1,1)	(0,1)	(0,0)	1
(1,1)	(1,1)	(1,0)	1