Circuits combinatoires et logique booléenne

Thèmes architectures matérielles et types de données de base

Première NSI Lycée du Parc

Table des matières

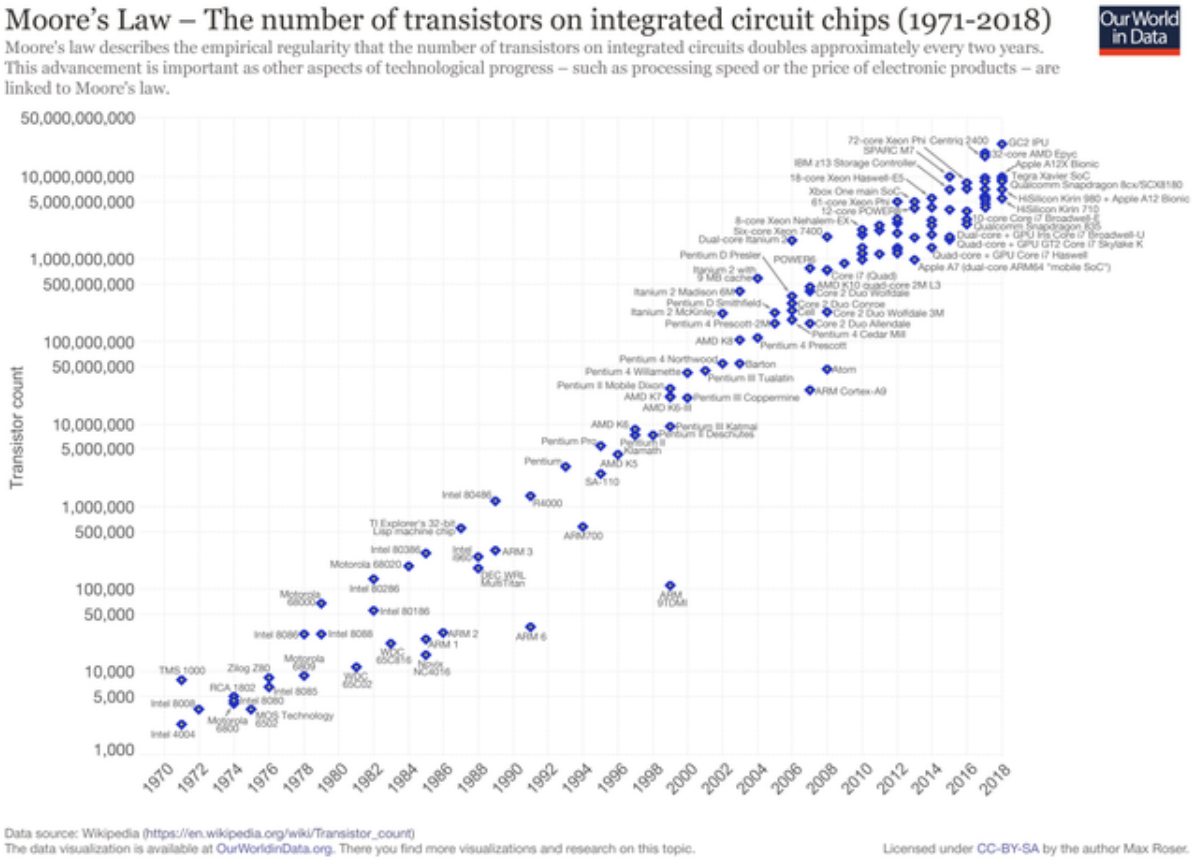
Crédits

Ce cours est largement inspiré du chapitre 22 du manuel NSI de la collection Tortue chez Ellipsen auteurs : Ballabonski, Conchon, Filliatre, N'Guyen.

Préambule

Les circuits d'un ordinateur manipulent uniquement des 0 ou des 1 représentés en interne par des tensions hautes ou basses. Les premiers ordinateurs construits dans la période 1945-1950 sont basés sur une technologie de tube à vide ou tube électrique. En 1947, aux laboratoires Bell, Shockley, Bardeen et Brattain inventent le **transistor** au *germanium* un petit composant électronique qui se comporte comme un interrupteur. Les transistors, plus petits et dissipant moins de chaleur, vont supplanter les tubes électriques : en 1954 le *germanium* est remplacé par le *silicium*, en 1955 apparaissent les premiers ordinateurs entièrement transistorisés, en 1960 le transistor à effet de champ permet l'intégration de dizaines composants dans un centimètre carré. Les transistors sont ensuite directement gravés dans une plaque de *silicium* constituant un **cicrcuit intégré**. En 1965 Gordon Moore futur directeur d'Intel énonce

la loi empirique portant son nom qui fixe une feuille de route à l'industrie des mircroprocesseurs : le doublement de la densité d'intégration des transistors tous les deux ans. Cette loi s'est vérifiée jusqu'à présent avec une finesse de gravure d'environ 5 nanomètres en 2020. Le graphique ci-dessous représente l'évolution du nombre de transistors par circuit intégré.



1 Portes logiques

1.1 Le transistor porte logique de base

Définition 1

Un **transistor** possède trois broches : la grille, la sortie (ou drain) et la source soumis à des états de tension haute ou basse qu'on peut assimiler aux valeurs binaires 1 et 0 d'un **bit**. Si la tension appliquée sur la grille est haute (bit à 1) alors le transitor laisse passer le courant entre la source d'énergie et la sortie et cette dernière passe à l'état de tension basse (bit à 0), sinon la sortie reste en tension haute (bit 1).

Une **fonction logique** prend un ou plusieurs bits en entrée et renvoie un ou plusieurs bits en sortie. Une **porte logique** est un circuit électronique représentant une **fonction logique**.

Une **table logique** représente les sorties produites par une fonction logique pour toutes les entrées

possibles.

Un transistor représente une fonction logique dont le bit d'entrée est l'état de tension de la grille et le bit de sortie, l'état de tension de la sortie. La **table logique** (table 1) associée est celle du **NON logique** ou **Inverseur**.

Fichier de test Logisim : transistor.circ.

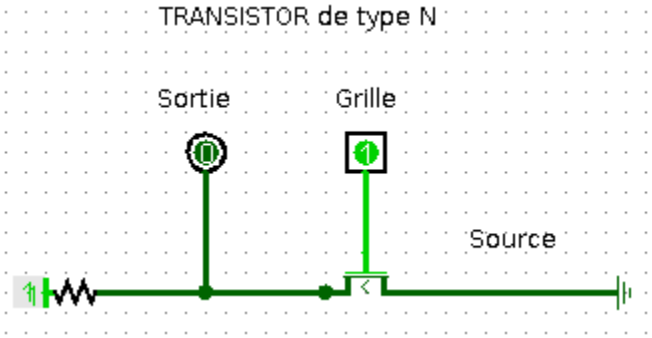
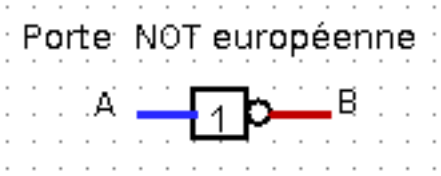
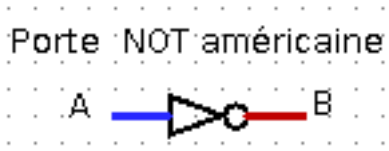


Table 1: **Table logique d'une porte NON**

A	B = NON(A)
0	1
1	0

Il existe deux conventions de représentation symbolique des portes logiques, une européenne et une américaine.





Tutoriel video Logisim : le transistor

1.2 D'autres portes logiques

1.2.1 Transistors en série ou en parallèle

Exercice 1

On donne ci-dessous les représentations de deux portes logiques :

- La **porte NAND** constituée de deux transistors en série

- La **porte NOR** constituée de deux transistors en parallèle

Chacune de ces portes logiques comportent deux bits d'entrée : A pour la grille du transistor 1 et B pour la grille du transistor 2 et un bit de sortie.

Compléter leurs tables logiques.

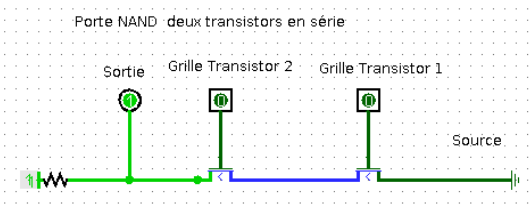
Vérifier avec Logisim et les fichiers porte_NAND.circ et porte_NOR.circ. Si Logisim n'est pas disponible, on pourra utiliser l'outil en ligne logic.ly

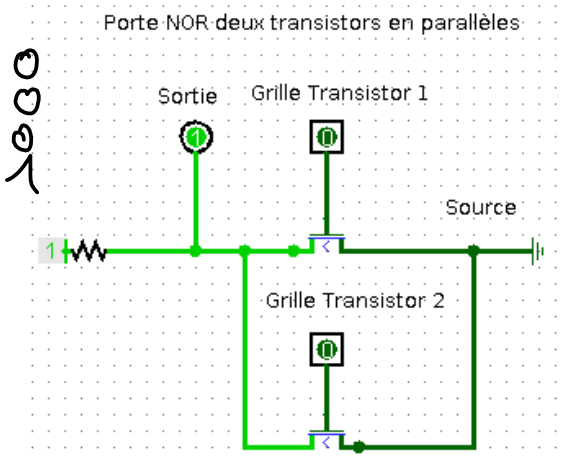
Tutoriel video Logisim : porte NAND

Tutoriel video Logisim : porte NOR

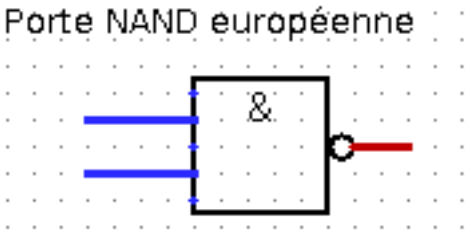
A	B	NAND(A, B)
0	0	
0	1	
1	0	
1	1	

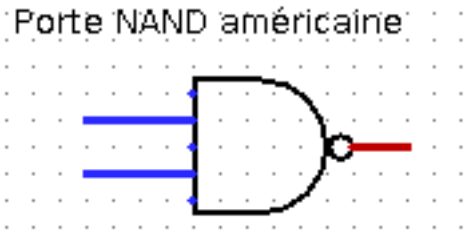
A	B	NOR(A, B)
0	0	1
0	1	1
1	0	1
1	1	0

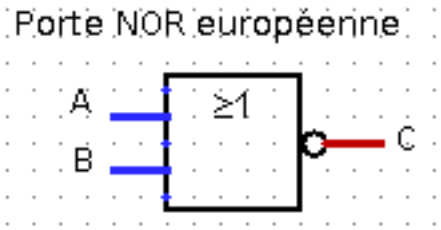


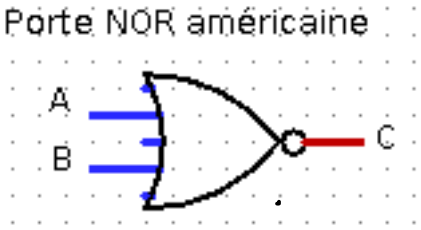


Voici les représentations symboliques des portes logiques NAND et NOR :







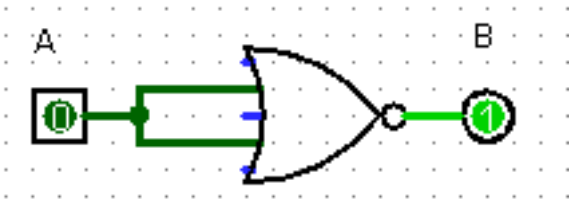


1.2.2 Portes logiques et fonctions logiques élémentaires

Exercice 2

Fichier de test Logisim : exercice2.circ.

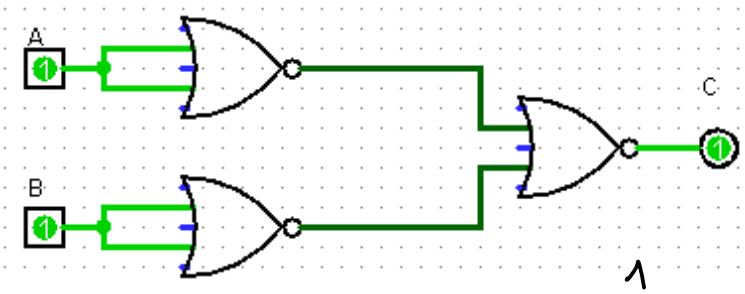
1. Compléter la table logique de la porte logique représentée par le circuit ci-dessous. Quelle porte logique peut-on ainsi représenter ?



A	B = f(A)
0	
1	

2. Compléter la table logique de la porte logique représentée par le circuit ci-dessous. Quelle fonction logique correspond à cette porte logique ?

porte NON à partir d'une porte NOR



1 0

A	B	C = g(A, B)
0	0	
0	1	
1	0	
1	1	

porte ET à partir de 3 portes NOR

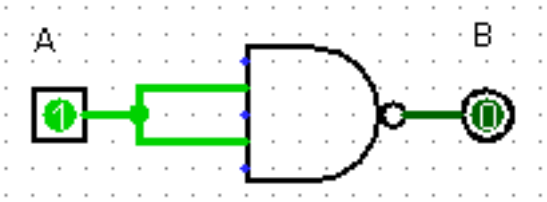
Tutoriel video Logisim : exercice 2

Exercice 3

Fichier de test Logisim : exercice3.circ.

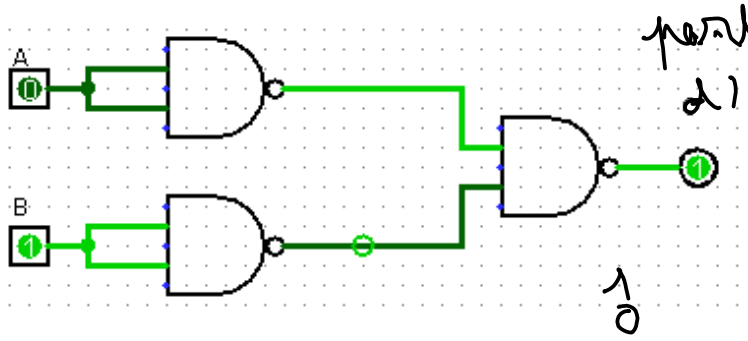
1. Compléter la table logique de la porte logique représentée par le circuit ci-dessous. Quelle porte logique peut-on ainsi représenter ?

0 0 0 1



A	B = f(A)
0	
1	

2. Compléter la table logique de la porte logique représentée par le circuit ci-dessous. Quelle fonction logique correspond à cette porte logique ?



porte NOT à partir d'une porte NAND

A	B	C = g(A, B)
0	0	
0	1	
1	0	
1	1	

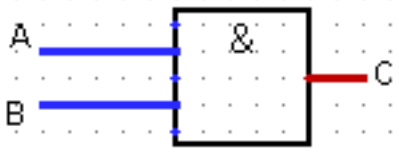
porte OR à partir de 3 portes NAND

Tutoriel video Logisim : exercice 3

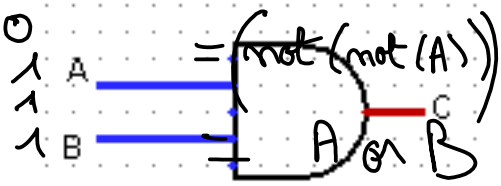
Voici les représentations symboliques des portes logiques `AND` et `OR` :

loi de Morgan.
not ((not A) and (not B))

Porte AND européenne



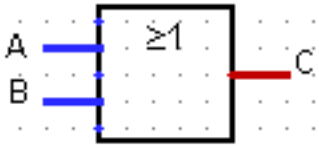
Porte AND américaine



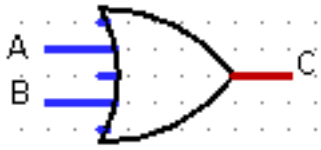
= (not(not(A))) or (not(not(B)))

A or B

Porte OR européenne



Porte OR américaine



Exercice 4

1. Construire un circuit représentant une porte `OR` uniquement avec des portes `NOR`.
2. Construire un circuit représentant une porte `AND` uniquement avec des portes `NAND`.

Ainsi chacune des portes, `NAND` ou `NOR` permet de construire les portes `NOT`, `OR`, `AND`. Toute porte logique pouvant s'exprimer à l'aide de ces trois portes, les portes `NAND` et `NOR` sont dites *universelles*.

Tutoriel video Logisim : exercice 4

2 Fonctions booléennes

2.1 Fonctions booléennes

Définition 2

- Un **booléen** est un type de données pouvant prendre deux valeurs `True` (Vrai) ou `False` (Faux) qu'on représente numériquement par un **bit** de valeur 1 pour `True` ou 0 pour `False`. Electroniquement, les valeurs 1 et 0 se traduisent respectivement par des tensions haute ou basse.
- Une **fonction booléenne** f associe un booléen à un ou plusieurs booléens.
- Une **fonction booléenne** avec n arguments est définie sur un ensemble $\{0;1\}^n$ à 2^n valeurs et prend ses valeurs dans $\{0;1\}$ qui a 2 éléments. On peut recenser les 2^n évaluations d'une fonction booléenne à n arguments dans une **table de vérité** qui la définit entièrement. Il existe 2^{2^n} fonctions booléennes à n arguments.
- Une **porte logique** est la représentation sous forme de circuit d'une fonction booléenne et sa **table logique** est la **table de vérité** de cette fonction.

Exercice 5

1. Compléter la fonction `Python` ci-dessous pour qu'elle affiche la table de vérité d'une fonction booléenne à deux entrées. Expliquer le rôle de la fonction `int`.

```
def table_verite_2bits(fonction):
    print('|{:^10}|{:^10}|{:^15}|'.format('a','b',fonction.__name__+'(a,b
        )'))
    for a in ..............:
        for b in ..............:
            print('|{:^10}|{:^10}|{:^15}|'.format(......, ......,
            int(fonction(bool(a),bool(b)))))
```

2. Vérifier que les tables de vérité affichées pour les fonctions `bool.__or__`, `bool.__and__` et `bool.__not__` sont correctes.

```
In [4]: table_verite_2bits(bool.__or__)
```

```
|   a   |   b   |   __or__(a,b)   |
|   1   |   1   |        1        |
|   1   |   0   |        1        |
|   0   |   1   |        1        |
|   0   |   0   |        0        |
```

Tutoriel video : exercice 5

Propriété 1

On peut exprimer toute fonction booléenne à l'aide de trois fonctions booléennes élémentaires :

- La *négation* de x est une fonction à 1 bit d'entrée (unaire) notée $\neg x$ ou $\overline{x}$.
  Si `x` est un booléen, sa *négation* est `not x` en `Python`.

x	$\neg x$
0	
1	

- La *conjonction* de x et y est une fonction à 2 bits d'entrée (binaire) notée $x \wedge y$ ou $x.y$.
  Si `x` et `y` sont des booléens, leur *conjonction* est `x and y` en `Python`.

x	y	$x \wedge y$
0	0	
0	1	
1	0	
1	1	

- La *disconjonction* de x et y est une fonction à 2 bits d'entrée (binaire) notée $x \vee y$ ou $x + y$.
  Si `x` et `y` sont des booléens, leur *disjonction* est `x or y` en `Python`.

x	y	$x \vee y$
0	0	
0	1	
1	0	
1	1	

Propriété 2

1. Les fonctions booléennes élémentaires respectent un certain nombre de règles qui permettent de simplifier les expressions booléennes complexes :

- *opérateur involutif* : $\neg(\neg x) = x$ et $\overline{\overline{x}} = x$
- *élément neutre* : $1 \wedge x = x$ et $1.x = x$ ou $0 \vee x = x$ et $0 + x = x$
- *élément absorbant* : $0 \wedge x = 0$ et $0.x = 0$ ou $1 \vee x = x$ et $1 + x = 1$
- *idempotence* : $x \wedge x = x$ et $x.x = x$ ou $x \vee x = x$ et $x + x = x$
- *complément* : $x \wedge (\neg x) = 0$ et $x.(\overline{x}) = 0$ ou $x \vee (\neg x) = 1$ et $x + \overline{x} = 1$
- *commutativité* : $x \wedge y = y \wedge x$ et $x.y = y.x$ ou $x \vee y = y \vee x$ et $x + y = y + x$
- *associativité* : $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ et $x.(y.z) = (x.y).z$ ou $x \vee (y \vee z) = (x \vee y) \vee z$ et $x + (y + z) = (x + y) + z$
- *distributivité* : $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ et $x.(y+z) = x.y + x.z$ ou $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ et $x + (y.z) = (x + y).(x + z)$
- *loi de Morgan* : $\neg(x \wedge y) = \neg x \vee \neg y$ et $\overline{x.y} = \overline{x} + \overline{y}$ ou $\neg(x \vee y) = \neg x \wedge \neg y$ et $\overline{x + y} = \overline{x}.\overline{y}$

2. Les fonctions booléennes élémentaire respectent des règles de priorité : la *négation* est prioritaire sur la *conjonction* qui est prioritaire sur la *disjonction*.
Il est recommandé de mettre des parenthèses plutôt que d'appliquer les règles de priorité dans l'écriture des expressions booléennes.

2.2 QCM types E3C

Exercice 6

1. Parmi les quatre expressions suivantes, laquelle s'évalue en True ?

- **Réponse A :** `False and (True and False)`
- **Réponse B :** `False or (True and False)`
- **Réponse B :** `True and (True and False)`
- **Réponse C :** `True or (True and False)`

2. Sachant que l'expression `not(a or b)` a la valeur `True`, quelles peuvent être les valeurs des variables booléennes a et b ?

- **Réponse A :** `True` et `True`
- **Réponse B :** `False` et `True`
- **Réponse C :** `True` et `False`
- **Réponse D :** `False` et `False`

3. Pour quelles valeurs booléennes des variables `a, b` et `c` l'expression `(a or b)and (not c)` a-t-elle pour valeur `True`

- **Réponse A :** `a = True b = False c = True`
- **Réponse B :** `a = True b = False c = False`

- **Réponse C :** `a = False b = False c = True`
- **Réponse D :** `a = False b = True c = True`

4. Si A et B sont des variables booléennes, laquelle de ces expressions booléennes est équivalente à `(not A)or B` ?

- **Réponse A :** `(A and B)or (not A and B)`
- **Réponse B :** `(A and B)or (not A and B)` = `(not A and not B)`
- **Réponse C :** `(not A and B)or (not A and not B)`
- **Réponse D :** `(A and B)or (not A and not B)`

5. Choisir une expression booléenne pour la variable S qui satisfait la table de vérité suivante.

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

- **Réponse A :** A ou (non B)
- **Réponse B :** (non A) ou B
- **Réponse C :** (non A) ou (non B)
- **Réponse D :** non (A ou B)

6. On considère une formule booléenne form des variables booléennes `a` et `b` dont voici la table de vérité.

a	b	form
True	True	False
False	True	False
True	False	True
False	False	False

Quelle est cette formule booléenne ?

- **Réponse A :** `a and b`
- **Réponse B :** `a or b`
- **Réponse C :** `a and not(b)`
- **Réponse D :** `not(a) or b`

Tutoriel video : exercice 6

3 Circuits combinatoires

3.1 Définition

Définition 3

Un **circuit logique combinatoire** permet de réaliser une ou plusieurs fonctions booléennes : ses sorties ne dépendent que de l'état actuel de ses entrées. Les portes logiques `NOT`, `NOR`, `NAND`, `AND`, `OR` et `XOR` sont des circuits combinatoires.

Il existe d'autres circuits, dits séquentiels, dont les sorties se calculent non seulement à partir de leurs valeurs d'entrée actuelles mais aussi à partir de leurs états précédents : le facteur temps intervient. Ils utilisent des circuits de mémoire pour mémoriser leurs états antérieurs.

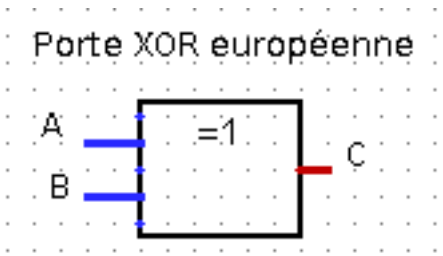
Exercice 7

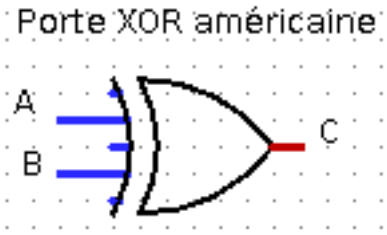
On considère la fonction booléenne dont la table de vérité est :

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0

1. Exprimer chacune des lignes où la fonction prend la valeur 1 comme la *conjonction* des entrées en remplaçant chaque 1 par la variable qu'il représente et chaque 0 par la négation de la variable. Par exemple le 1 de la deuxième ligne s'écrira $\bar{x}.y$.
2. On peut alors écrire $f(x,y)$ comme la *disjonction* des *formes conjonctives* obtenues à la question précédente. En déduire une expression booléenne de $f(x,y)$.
3. Ouvrir le logiciel Logisim et construire une porte logique représentant cette fonction booléenne.
4. Cette fonction s'appelle `OU EXCLUSIF` ou `XOR`. Ce nom vous paraît-il bien choisi ?

Voici les représentations symboliques de la porte logique `XOR` :





3.2 Demi-additionneur et additionneur 1 bit

Exercice 8

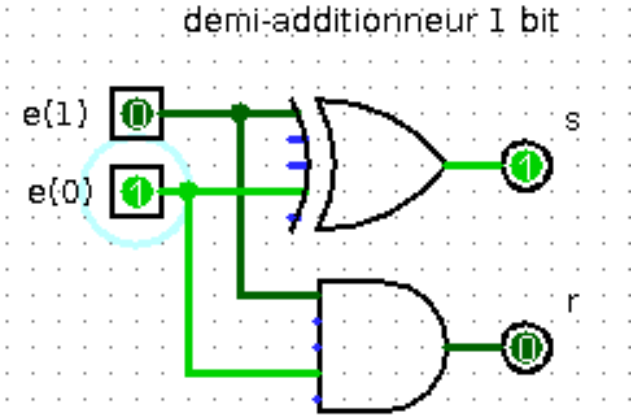
1. Effectuer les additions binaires : $0+0$, $0+1$, $1+0$ et $1+1$.
2. Un **demi-additionneur binaire 1 bit** est un circuit combinatoire qui possède :
   - deux entrées : deux bits d'opérande e_0 et e_1 ;
   - deux sorties : un bit de résultat s et un bit de retenue sortante r.

La sortie s prend pour valeur le bit des unités et la sortie r le bit de retenue sortante, lorsqu'on additionne les deux bits d'entrée e_0 et e_1.

1. Compléter la table de vérité de ce circuit combinatoire :

e_0	e_1	s	r
0	0		
0	1		
1	0		
1	1		

4. Justifier qu'un **demi-additionneur binaire 1 bit** peut être représenté par le circuit ci-dessous.



5. Ouvrir le logiciel Logisim et construire un circuit combinatoire représentant un **demi-additionneur binaire 1 bit**.

Tutoriel video : exercice 8

Exercice 9

Un **additionneur binaire 1 bit** est un circuit combinatoire qui possède :

- trois entrées : deux bits d'opérande e_0 et e_1 et un bit de retenue entrante r_0
- deux bits de sortie : un bit de résultat s_2 et un bit de retenue sortante r_3.

(A and B) or (not A and B) or (not A and not B)

(A and B) or (not A and not B)

A	B	(not A) or B	B or (not A and not B)	(A and B) or (not A and not B)
0	0	1	1	1
0	1	1	1	0
1	0	0	0	
1	1	1	1	

$$B \text{ or } ((\text{not } A) \text{ and } (\text{not } B))$$

$$= (B \text{ or } (\text{not } A)) \text{ and } (B \text{ or } (\text{not } B))$$

$$= B \text{ or } (\text{not } A)$$

1. Compléter les colonnes de la table de vérité d'un **additionneur binaire 1 bit** pour le bit de résultat s_2 et le bit retenue sortante r_3.

e_0	e_1	r_0	$s_1 = \ldots\ldots$	$r_1 = \ldots\ldots$	$s_2 = \ldots\ldots$	$r_2 = \ldots\ldots$	$r_3 = \ldots\ldots$
0	0	0					
0	1	0					
1	0	0					
1	1	0					
0	0	1					
0	1	1					
1	0	1					
1	1	1					

2. Un **additionneur binaire 1 bit** peut être réalisé à l'aide de deux **demi-additionneurs binaires 1 bit** :

   - Le premier **demi-additionneur binaire 1 bit** prend en entrée les bits d'opérande e_0 et e_1 et retourne en sortie un bit de résultat intermédiaire s_1 et un bit de retenue sortante intermédiaire r_1. Donner une expression booléenne de s_1 et r_1 en fonction de e_0 et e_1.
   - Le second **demi-additionneur binaire 1 bit** prend en entrée le bit de résultat s_1 et le bit de retenue entrante r_0 et retourne en sortie le bit de résultat final s_2 et un bit de retenue sortante intermédiaire r_2. Donner une expression booléenne de s_2 et r_2 en fonction de s_1 et r_0.
   - Enfin, la retenue sortante r_3 s'obtient à partir de la retenue sortante r_1 du premier demi-additionneur et de la retenue sortante r_2 du second. Donner une expression booléenne de r_3 en fonction de r_1 et r_2.

   Compléter les colonnes s_1, r_1 et r_2 puis s_2 et r_3 de la table de vérité de l'**additionneur binaire à 1 bit**.

3. Avec le logiciel Logisim ouvrir le fichier contenant le demi-additionneur de l'exercice précédent.

   - Ajouter un nouveau circuit avec `Add a circuit` , le nommer `additionneur1bit` puis copier/coller dedans le circuit du **demi-additionneur binaire 1 bit**. Compléter le circuit pour obtenir un **additionneur binaire 1 bit**.
   - Ajouter un nouveau circuit avec `Add a circuit` , le nommer `additionneur2bits` puis copier/coller dedans le circuit de l' **additionneur binaire 1 bit**. Compléter le circuit pour obtenir un **additionneur binaire 2 bits**.

Tutoriel video : exercice 9